浙江大学

课程设计报告

中文题目：基于 FPGA 的吃豆人游戏设计

英文题目：Pacman Game Design Based on FPGA

课　　程：数字逻辑设计

任课教师：刘海风

指导教师：郭家豪、洪奇军

组　　长：Chen Tao

组　　员：cht、牢靳

日　　期：2024 年 6 月 12 日

摘要

本次课程设计旨在开发一款基于现场可编程门阵列（FPGA）的吃豆人游戏。通过运用数字逻辑设计的原理和 Verilog 硬件描述语言，我们成功实现了游戏的核心逻辑和外设接口。本设计采用了模块化的方法，将游戏内容、逻辑和外设使用明确区分，以便于开发和调试。

在游戏逻辑方面，我们实现了吃豆人的基本移动控制、得分逻辑以及与鬼魂的互动。特别地，针对吃豆人与鬼魂的碰撞检测和随机路径选择，我们设计了一套有效的算法，增强了游戏的可玩性和挑战性。外设使用方面，我们通过 VGA 屏幕实现了游戏图形的显示，利用蜂鸣器播放音乐和音效，以及使用 PS/2 键盘进行玩家输入。

在实现过程中，我们面临了包括时序违例、显示噪点和多驱动问题在内的多项技术挑战。通过使用最新版的 Vivado 工具和优化设计方法，我们逐一解决了这些问题。此外，我们还探索了系统 Verilog 语言的高级特性，以提高设计的灵活性和效率。

最终，我们完成了游戏的设计和实现，并在 Sword Kintex 7 实验平台上进行了测试。测试结果表明，游戏运行流畅，玩家可以控制吃豆人收集豆子，躲避鬼魂，享受游戏带来的乐趣。本项目不仅加深了我们对 FPGA 和数字逻辑设计的理解，而且提升了我们解决实际问题的能力。

关键词：System Verilog；Vivado；FPGA；数字逻辑设计；吃豆人游戏；模块化

目录

1 绪论

1.1 游戏介绍

吃豆人是一款经典的街机游戏。该游戏的背景以黑色为主。在游戏中，玩家控制一个黄色的圆形角色“吃豆人”，通过操作方向键让吃豆人在迷宫中吃掉所有豆子，并躲避鬼魂的追击。游戏简单易上手，但也需要玩家灵活应对，具有一定的挑战性和趣味性。

1.2 硬件描述语言

Verilog 是一种专用于电子系统设计自动化的硬件描述语言，它允许设计者以高层次的抽象来描述、设计和模拟数字电路。这种语言特别适用于集成电路和可编程逻辑设备的设计。Verilog 的核心优势在于其模块化设计，使得复杂的电路可以被分解为更小、更易于管理的模块。它支持并发性描述，允许设计者表达电路中同时发生的多个操作，并通过仿真来验证电路的行为。

Verilog 的语法与 C 语言相似，这为有编程背景的设计者提供了便利，使其能够更快地掌握和使用。它还提供了强大的测试和验证机制，使得设计者能够在实际硬件制造之前，通过软件模拟来确保电路的功能和时序正确性。此外，Verilog 代码可以被综合成更具体的门级电路实现，这是将设计转化为实际硬件的关键步骤。

Verilog 的设计流程通常包括编写代码、进行仿真测试、综合、布局与布线，最终实现硬件。这一流程得到了多种商业和开源工具的支持，包括仿真器、综合器和调试工具，这些工具进一步增强了 Verilog 在数字电路设计中的实用性。作为 IEEE 1364 标准的一部分，Verilog 保证了不同工具和平台之间的兼容性，使其在电子、通信、计算机科学和自动化等多个领域得到了广泛应用。总的来说，Verilog 是一种功能强大、灵活且广泛支持的硬件描述语言，是数字电路设计中不可或缺的工具。

值得一提的是，Verilog 并不支持模块的向量数组输入输出，所以我们在实际设计时更多实用 System Verilog 语言。Verilog 是 System Verilog 的子集，System Verilog 支持所有 Verilog 语法，所以我们可以继续使用 Verilog 语法，没有遇到工程迁移的问题。

1.3 设计工具

- 软件
  - Vivado 2022.2、Vivado 2024.1。Vivado 是 Xilinx 公司开发的一款集成开发环境（IDE），专门用于设计、仿真和实现 Xilinx FPGA、SoC 和 3D ICs 的数字电路。
  - Photoshop。用于绘制背景图、图标和动画静帧。
  - http://onlinesequencer.net。在线 midi 编辑器，用于设计音效。
- 硬件
  - 计算机。
  - Sword Kintex 7 实验平台。
  - 外接 VGA 显示器。
  - PS/2 键盘。

1.4 主要难点

- 外设使用
  - VGA 显示原理学习，显示时序分析。
  - ip 核配置学习。
  - 音乐音效设计（主要是乐理知识有限）。
- 游戏逻辑
  - 吃豆人与鬼魂根据地图的碰撞判定，特别是吃豆人的连续运动使判断尤为复杂
  - 鬼魂根据每次撞墙后的对于地图四周的判断与对方向的选择

2 基于 FPGA 的吃豆人游戏设计方案

2.1 主要结构

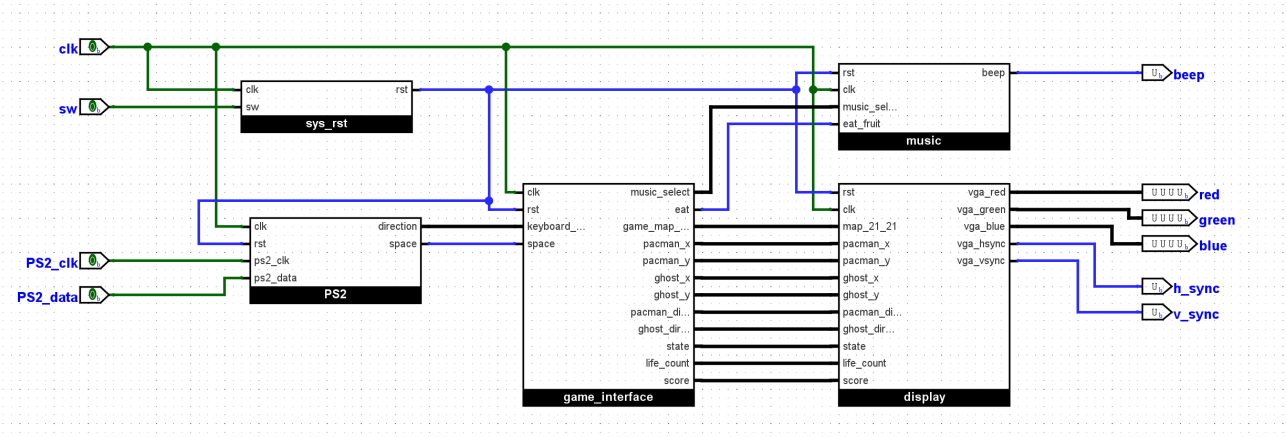

图 1　顶层模块电路原理图

图 1 展示了我们的顶层模块设计。我们将游戏内容和逻辑集成在`game_interface`核心模块中，为输入输出设计的单独的模块。其中，`PS2`模块接受来自键盘的信号，将输入转换为编码过的方向信号传给`game_interface`模块。`display`模块读取游戏中所有需要显示的状态，进行画面渲染和 VGA 的驱动。`music`模块接受来自`game_interface`模块的音乐选择编码，驱动蜂鸣器播放相应的音乐或音效。`sys_rst`模块负责上电重置并统一管理`rst`信号。

我们的模块化设计遵循了标准的设计原则，在前期为输入输出定义、需求对齐提供了极大便利；在中期让代码编写和测试的分工合作更加顺利；在后期也方便 debug 和成果展示的进行。

2.2 游戏逻辑

2.2.1 吃豆人逻辑

吃豆人在游戏开始 4.3 秒之后左上角允许开始移动，键盘的上下左右键分别控制方向，值得注意的是，一次按键按下之后方向会一直保持，直到下一个不同的方向键按下才会改变。为了保持游戏平衡，吃豆人的移动速度略微小于鬼的速度。

吃豆计分逻辑：当豆人触碰到任然存在的豆子的情况下，豆子将会消失，游戏积分加一。值得注意的是，由于吃豆人的胃口有限，同一时间内豆人不管吃一个还是两个豆子，分数只会加一。

当豆人与鬼有部分接触时，豆人的生命值将会减一，如果豆人的生命值为 0，豆人将会回到原位。游戏结束。

2.2.2 鬼魂逻辑

鬼魂在游戏开始 4.3 秒之后会从地图中间瞬移到地图中上位置，接着朝着一个方向移动。鬼魂的移动遵循以下规则：当鬼魂触碰到墙壁之前，它将保持在同一方向移动；当鬼魂预测的下一个位置有墙体存在时，他将会根据下一个本位置的其他方向是否有墙体进行一个随机数判断，即随机朝一个方向转向并前进。

当鬼的移动方向正前方探测到吃豆人的存在之后，游戏音乐将会变得急促。

当鬼触碰到玩家时，鬼将会回到出生点，并按以上逻辑继续移动。

当玩家的生命值为 0 时，鬼将回到起始点，此时，鬼会局促不安地在起始地快速移动。

2.3 外设使用

2.3.1 VGA 屏幕显示

2.3.1.1 VGA 显示原理

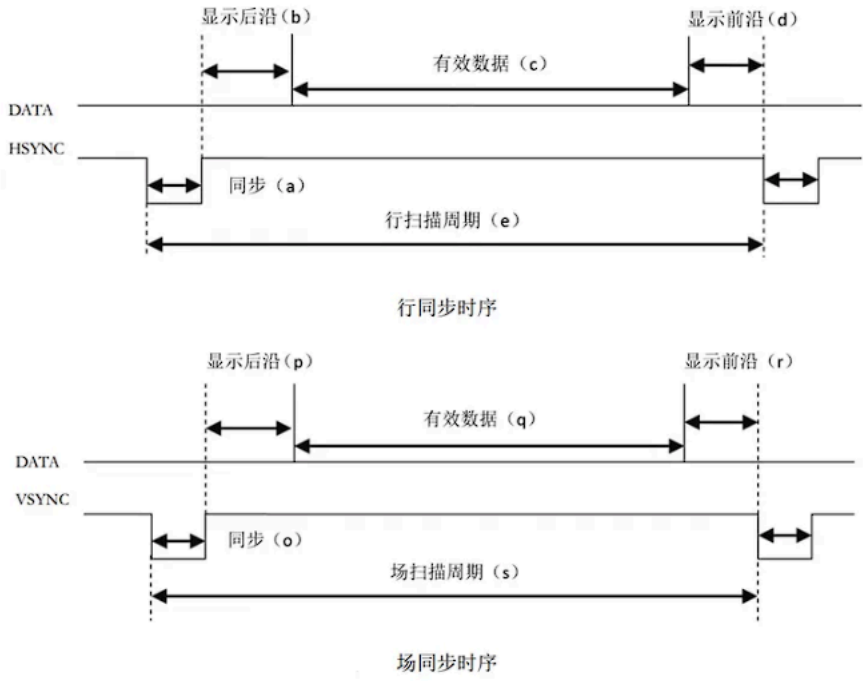

图 2　VGA 显示时序

VGA 协议通过每一帧扫描屏幕上的所有像素点来实现显示，其时序如图 2 所示，显示器依赖行列扫描信号与显示模块实现同步，每个阶段具体持续时间时间参考文档。我们直接使用文档提供的`vgac`模块来实现 VGA 显示。

分析`vgac`代码。`vgac`模块使用计数器控制`hs`和`vs`扫描信号，并提前一个`vga_clk`周期返回下一个周期将要显示的像素的`row_addr`和`col_addr`。同时，`vgac`将本周期输入的`d_in`拆分成`r`,`g`,`b`三种颜色的亮度数据，输出给显示器。

由此，我们可以设计一个`renderer`渲染器模块与`vgac`进行交互，每个周期返回`row_addr`和`col_addr`对应的像素点颜色信息`d_in[11:0]`。同时`renderer`模块需要实例化一系列 ROM ip 核来存储需要显示的图片素材，也需要实时获取游戏数据。

2.3.1.2 `display`模块设计

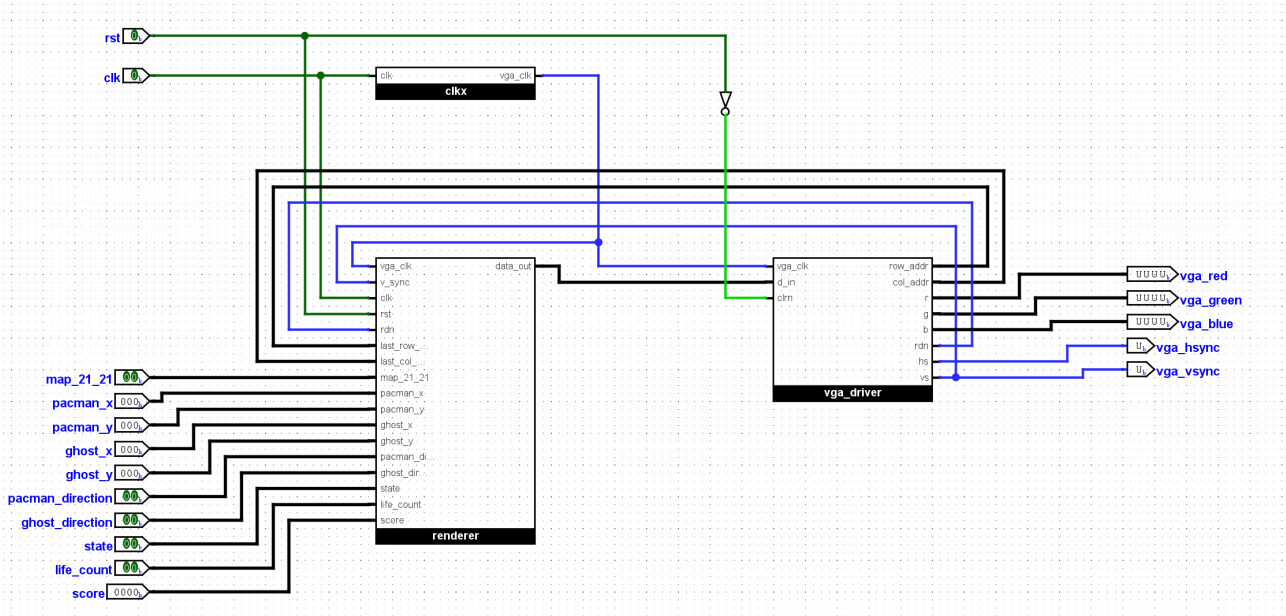

图 3 `display` 模块电路原理图

图 3 展示了 `display` 模块的接线。

其中，`renderer` 模块是渲染器，输入游戏的所有显示信息和 `vga_driver` 返回的 `row_addr`, `col_addr`，对于将要显示的像素点，在模块内找到正确的 ROM ip 核，计算正确的 ROM 地址并取出 12 位的像素颜色信息，输出给 `vga_driver` 模块。

`vga_driver` 模块直接使用了实验文档中提供的 `vgac` 模块的代码，管理 VGA 的行列扫描信号，将输入的像素颜色信息分配给 `vga_red`, `vga_green`, `vga_blue` 三个通道。

`clkx` 模块使用 Clock Wizard ip 核生成 25.175MHz 的 `vga_clk`，以驱动 `vga_driver` 模块。

2.3.1.3 图像素材绘制

使用 Photoshop 创建大小合适的画布，使用铅笔工具逐像素绘制图像。

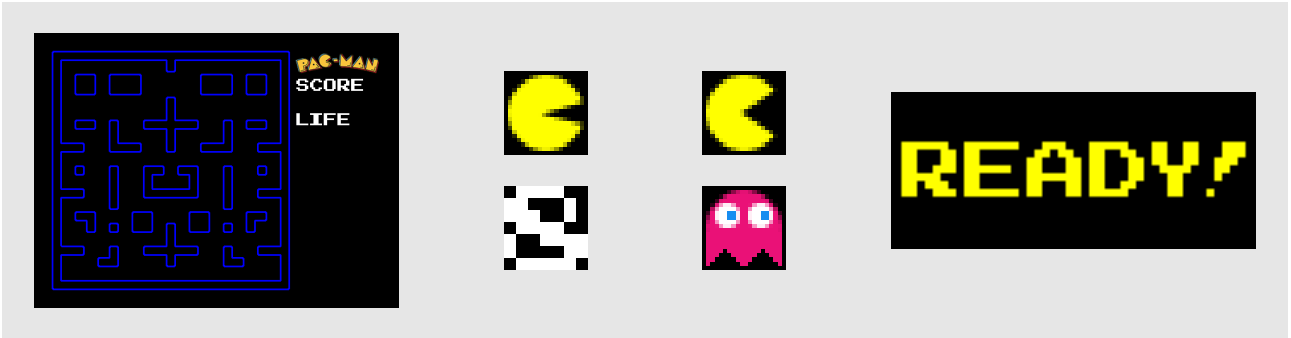

图 4 部分图像素材展示

图 4 展示了部分的图像素材。图像素材包括了背景、动画静帧和静态图标等。

2.3.1.4 coe 文件准备

我编写 python 程序（见于附录`png_convertor.py`）将 Photoshop 导出的 8-bit `.png` 格式图像转化为 4-bit `.coe` 格式的文件。直接读取图像的每个像素，将其颜色信息映射到 4-bitRGB 空间，并输出到 `.coe` 文件即可。

为了减少 ip 核的数量，我将每个角色的动画放在一个 ROM 中，将所有的数字放在一个 ROM 中，所以也需要将 `.coe` 文件进行相应合并。

2.3.1.5 ROM ip 核创建

Vivado 提供了片上内存的 ip 核 Block Memory Generator，我使用这个 ip 核来存储图片素材。

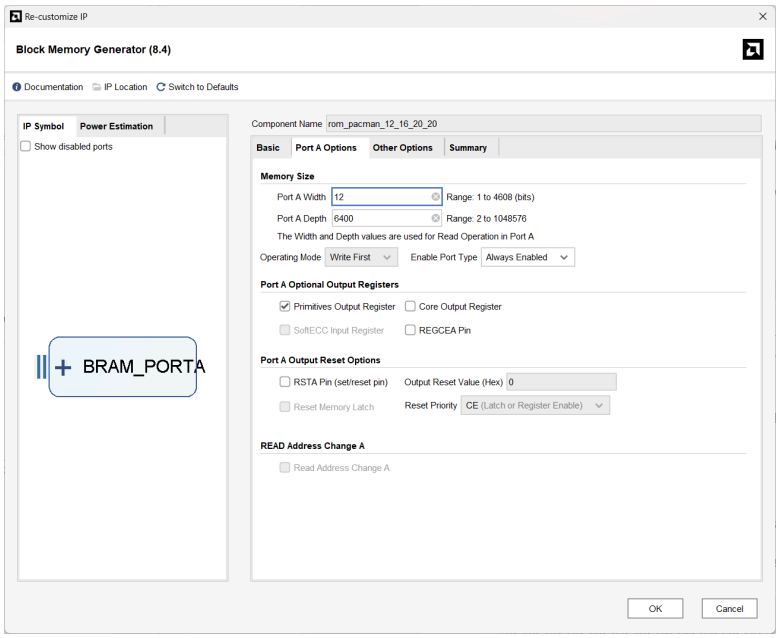

图 5　使用 Black Memory Generator 生成 ROM ip 核

首先在`Basic`标签页选择`Single Port ROM`，即单端口只读存储。然后在`Options`标签页选择`Always Enabled`，并按照需要存储的数据量大小填写位宽和位深，如图 5 中的`rom_pacman_12_16_20_20`需要存储 16 张 20*20 的 4-bit 图像，所以位深为 $16 \times 20 \times 20 = 6400$。最后在`Other Options`标签页选择加载初始化`.coe`文件，然后生成 ip 核即可。

调用 ip 核时需要注意 ip 核的端口定义如下：

```
ENTITY rom_pacman_12_16_20_20 IS
  PORT (
    clka : IN STD_LOGIC;
    addra : IN STD_LOGIC_VECTOR(12 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(11 DOWNTO 0)
  );
END rom_pacman_12_16_20_20;
```

2.3.2 蜂鸣器

2.3.2.1 蜂鸣器工作原理

板上蜂鸣器只有一个输入引脚，通过调节输入信号的频率来得到对应频率的声音输出，中仅实现最简单的方波音色。

参考十二平均律，我们计划使用从 C3 到 B6 共 48 个音高来完成音乐播放，频率对照如下：

```
integer note_freq [0:47] = {
  // C,   C#,  D,   D#,  E,   F,   F#,  G,   G#,  A,   A#,  B
    131, 139, 147, 156, 165, 175, 185, 196, 208, 220, 233, 247, // C3 - B3
    262, 277, 294, 311, 330, 349, 370, 392, 415, 440, 466, 494, // C4 - B4
    523, 554, 587, 622, 659, 698, 740, 784, 831, 880, 932, 988, // C5 - B5
    1047, 1109, 1175, 1245, 1319, 1397, 1480, 1568, 1661, 1760, 1865, 1976  // C6 - B6
};
```

在具体实现时，根据当前音符编码取用对应的频率，然后向蜂鸣器输出这个频率的方波信号即可。

2.3.2.2 `music` 模块设计

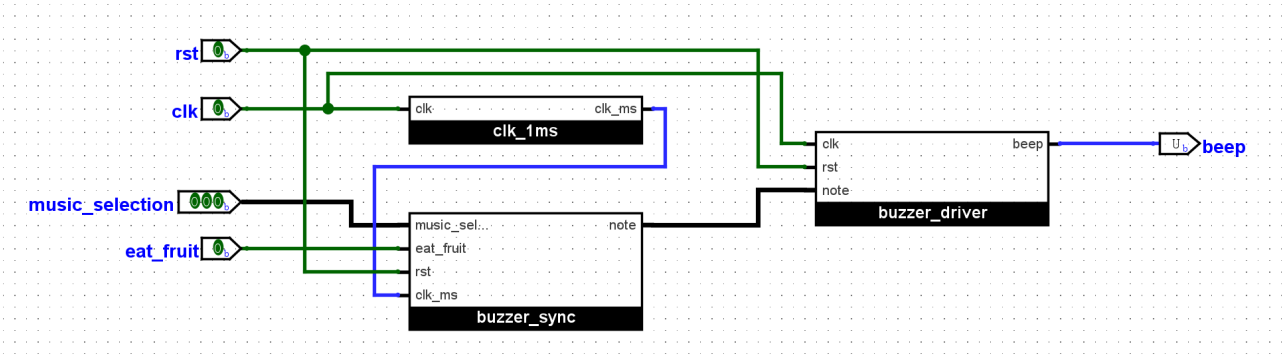

图 6 `music` 模块电路原理图

图 6 是 `music` 模块的原理图。

其中 `buzzer_driver` 存储了音高对照表，根据输入的一个 `note` 的对应音高对 `clk` 实现分频，进行 `beep` 信号的翻转，从而输出音高频率的方波。

`buzzer_sync` 模块存储了游戏的所有音乐和音效，包括 Startup、Slow、Fast、Gameover 四种音乐以及 eat 音效。

2.3.2.3 音效设计

结合从互联网上找到的简谱资源以及我基础的乐理知识，我使用 Online Sequence 这个在线 midi 编辑器进行了音效设计。由于蜂鸣器不适合播放多轨音效，我只保留了主旋律和部分低音部分，并保证音符不重叠，并设置了合适的 BPM。

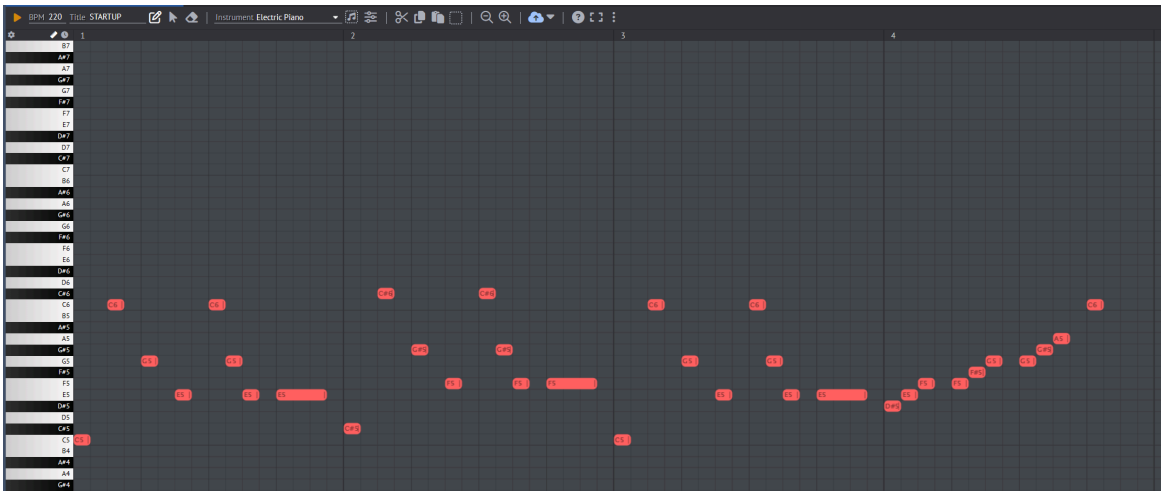

图 7　Startup 音乐设计

我采用 1ms 为音乐播放的最小分辨率，例如，图 7 中展示的 Startup 音乐为 220BPM，4/4 拍，粗略计算可以得知每个 16 分音符的持续时间为 68ms。

由于音乐所需的存储空间并不大，我使用 `integer` 数组存储音乐，并使用宏定义表示音符编码，例如 Startup 音乐的存储和表示如下：

```
// start up music
integer startup_time = 4352; // ms
integer startup_note_time = 68; // ms
integer startup [0:63] = {
    `C5, `empty,  `C6, `empty, `G5, `empty, `E5, `empty, `C6, `G5, `E5, `empty, `E5, `E5, `E5, `empty,
    `C_5, `empty, `C_6, `empty, `G_5, `empty, `F5, `empty, `C_6, `G_5, `F5, `empty, `F5, `F5, `F5, `empty,
    `C5, `empty, `C_6, `empty, `G5, `empty, `E5, `empty, `C6, `G5, `E5, `empty, `E5, `E5, `E5, `empty,
    `D_5, `E5, `F5, `empty, `F5, `F_5, `G5, `empty, `G5, `G_5, `A5, `empty, `C6, `empty, `empty, `empty
};
```

2.3.3 PS/2 键盘

本模块读入键盘的时钟信号和数据信号，由于键盘时钟信号慢于系统时钟，所以需要额外两个寄存器来保存上一个键盘数据信息，从而能够在一个时钟周期里，通过比较不同寄存器的值来判断按键的输入。通过不同的输入信息，我们可以决定当前的 direction 以及 space 输出。

上下左右的完整的数据包括通码：E0 和数据信息，以及断码：E0、F0 和数据信息，而 space 的通码与断码则相对于前者没有 E0 这个数据。其中，一个完整的数据位包括 11 位，其中仅有 8 位是有效位。

3 基于 FPGA 的吃豆人游戏设计实现

3.1 游戏内容

3.1.1 game_interface 模块

在 game_interface 模块中，我们实现了通过键盘输入方向值（即 keyboard_input）改变吃豆人的移动方向，通过吃豆子实现分数 score 的增加，实现幽灵进行随机路径追捕吃豆人，根据不同情况（如游戏开始前、游戏中、追逐、结束）播放不同的音乐等功能。最终该模块

的输出将通过 top 模块中的 vga 模块以及 ip 核的调用实现人物、地图以及界面的显示。该模块的定义如下图所示。

game_interface 包括 3 个子模块，分别是 pacman_move(记录吃豆人坐标、方向、分数、创建地图)，ghost_move 模块（记录幽灵方向、坐标、游戏状态、音乐播放、吃否抓住吃豆人），以及 clk_1ms 时钟分频模块（游戏时钟采用 1ms）。

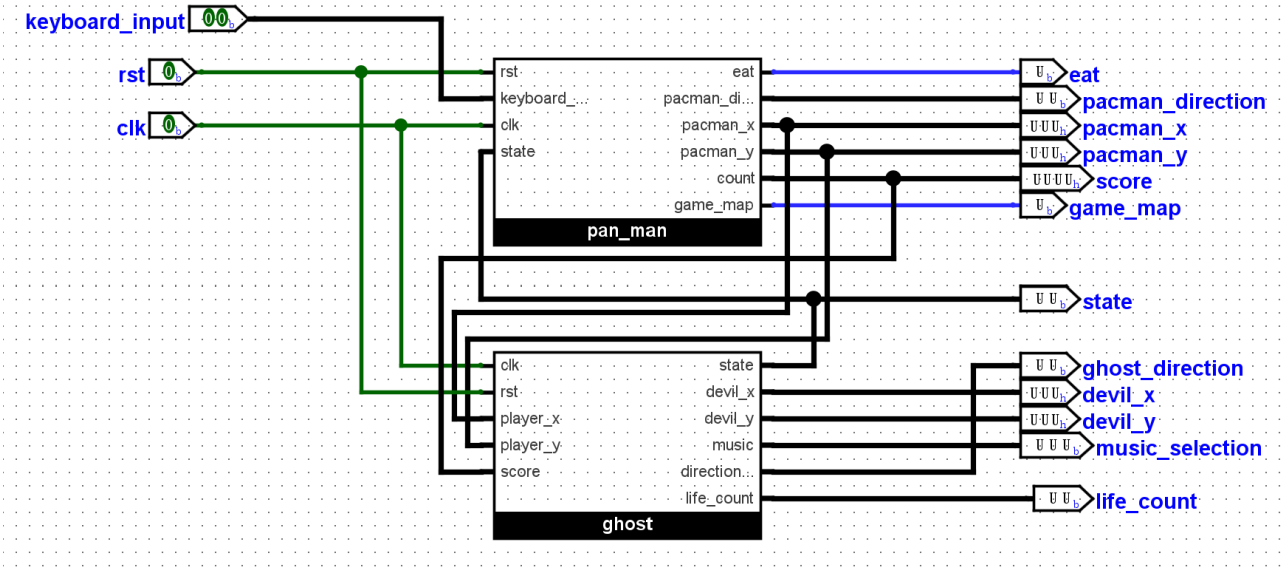

图 8 `game_interface` 模块电路原理图

3.1.2 地图实现

地图构建

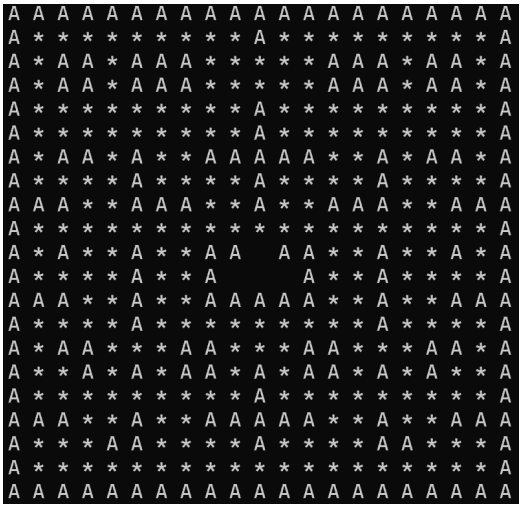

图 9 地图

3.1.3 吃豆人移动实现

这里我们定义上、下、左、右值分别为 0,1,2,3。当吃豆人接收到一个来自于 keyboard 的输入后，判断吃豆人移动方向。因为墙壁的存在，我们需要判断吃豆人在移动过程中是否会“撞墙”。由于我们的地图并不是离散的，因此不能简单地用吃豆人 x 和 y 坐标直接进行判断。这里我们采用“碰撞箱”进行碰撞模拟。

我们将 pacman 坐标抽象为左上角点，并将 9 位的 pacman_x 和 pacman_y 进行映射，映射方式为 pacman_x 和 pacman_y 对 20 取整（吃豆人像素大小为 20*20），映射后的 x 和 y 为 map_x 和 map_y。接着我们定义 counter_x 和 counter_y 为吃豆人另外的边界点。通过四个映射值我们可以判断吃豆人是否会撞墙，吃豆子。吃豆子时，eat 变成高电平，不管吃一个还是两个豆子，分数均加一，同时我们将 map 中对应的位置变成空白。以下是一个例子（以向上为例）

```
2'b00 : begin //up
     if(map[counter_y-1][map_x]==2'b01||map[counter_y-1][counter_x]==2'b01) begin //hit
               pacman_x <= pacman_x;
               pacman_y <= pacman_y; //don't change the x and y
               eat <= 0;
                                  // reg_direction <= 2'b00;
     end
     else begin
          if(map[counter_y-1][map_x]==2'b11&&map[counter_y-1][counter_x]==2'b00) begin
                map[map_y-1][map_x]=2'b00;
                eat <= 1;
                sum_of_dot=sum_of_dot-1;
          end
          else if(map[counter_y-1][map_x]==2'b00&&map[counter_y-1][counter_x]==2'b11) begin
          map[counter_y-1][counter_x]=2'b00;
          eat <= 1;
          sum_of_dot=sum_of_dot-1;
          end
          else if(map[counter_y-1][map_x]==2'b11&&map[counter_y-1][counter_x]==2'b11) begin
          if(map_x==counter_x) begin
               sum_of_dot=sum_of_dot-1;
          end
          else begin
               sum_of_dot=sum_of_dot-2;
          end
          map[counter_y-1][map_x]=2'b00;
          map[counter_y-1][counter_x]=2'b00;
          eat <= 1;
          // pacman_direction <= 2'b00;
          end
          pacman_x <= pacman_x;
          pacman_y <= pacman_y - 1;
          pacman_direction <= 2'b00;
          end
          // pacman_direction <= 2'b00;
end
```

代码仿真如下

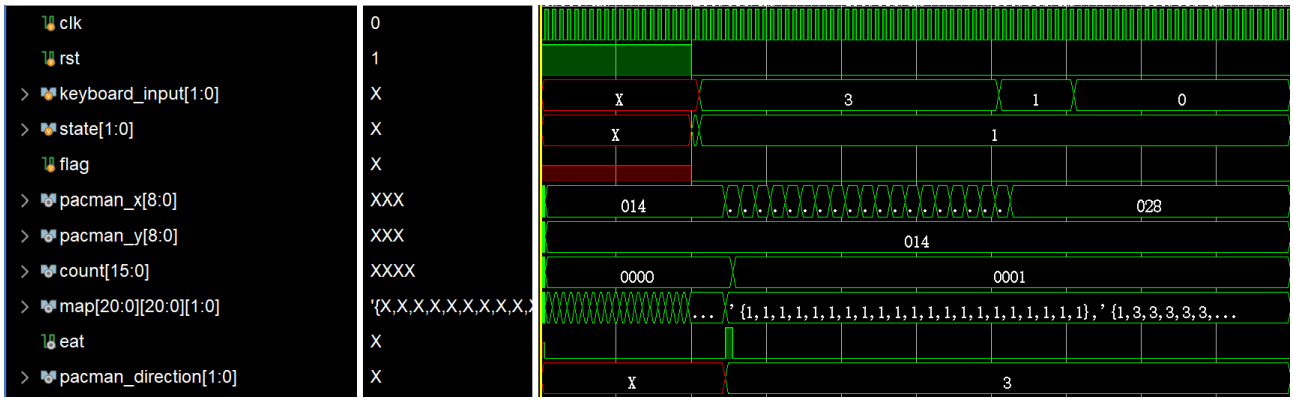

图 10 pacman 模块仿真

3.1.4 幽灵实现

3.1.4.1 随机数的实现

本模块采用伪随机数的选择，即在游戏重置时给予存储器 seed 一个初值；每次时钟上沿时，根据当前 score 得分并加以一系列数学运算改变其值。在选择转向时对随机数进行调用

```
if (1 && map[(devil_y/20)][(devil_x+19)/20-1]==1 && !(map[devil_y/20][(devil_x/20)+1]==1)) begin
    if (random % 3 == 0) begin
        devil_y <= devil_y + 1;
        direction_reg <= 1;
    end
    if (random % 3 == 1) begin
        devil_y <= devil_y - 1;
        direction_reg <= 0;
    end
    if (random % 3 == 2) begin
        devil_x <= devil_x + 1;
        direction_reg <= 3;
    end
end
else if (!(map[(devil_y+19)/20-1][devil_x/20]==1) && !(map[(devil_y/20)+1][devil_x/20]==1) && !
(map[(devil_y/20)][(devil_x+19)/20-1]==1) && map[devil_y/20][(devil_x/20)+1]==1) begin
    if (random % 3 == 0) begin
        devil_y <= devil_y + 1;
        direction_reg <= 1;
    end
    if (random % 3 == 1) begin
        devil_y <= devil_y - 1;
        direction_reg <= 0;
    end
    if (random % 3 == 2) begin
        devil_x <= devil_x - 1;
        direction_reg <= 2;
    end
end
else if (map[(devil_y+19)/20-1][devil_x/20]==1 && !(map[(devil_y/20)+1][devil_x/20]==1) && !
(map[(devil_y/20)][(devil_x+19)/20-1]==1) && map[devil_y/20][(devil_x/20)+1]==1) begin
    if (random % 2 == 0) begin
        devil_y <= devil_y + 1;
        direction_reg <= 1;
    end
    if (random % 2 == 1) begin
        devil_x <= devil_x + 1;
        direction_reg <= 3;
    end
end
else if (!(map[(devil_y+19)/20-1][devil_x/20]==1) && map[(devil_y/20)+1][devil_x/20]==1 && map[(devil_y/20)]
[(devil_x+19)/20-1]==1 && !(map[devil_y/20][(devil_x/20)+1]==1)) begin
    if (random % 2 == 0) begin
        devil_y <= devil_y - 1;
        direction_reg <= 0;
    end
    if (random % 2 == 1) begin
        devil_x <= devil_x + 1;
        direction_reg <= 3;
    end
end
else if (!(map[(devil_y+19)/20-1][devil_x/20]==1) && map[(devil_y/20)+1][devil_x/20]==1 && !
(map[(devil_y/20)][(devil_x+19)/20-1]==1) && map[devil_y/20][(devil_x/20)+1]==1) begin
    if (random % 2 == 0) begin
        devil_y <= devil_y - 1;
```

```
        direction_reg <= 0;
    end
    if (random % 2 == 1) begin
        devil_x <= devil_x - 1;
        direction_reg <= 2;
    end
end
```

3.1.4.2 幽灵抓住吃豆人

在本模块中，我们实现了幽灵的移动。幽灵在撞到墙之后会根据当前位置周围哪几个方向有墙壁，来进行方向的随机更换；“撞到墙“的判断类似于吃豆人的碰撞箱判断，此处不加以赘述。当幽灵的前进方向上有玩家视野的话，音乐将变得急促；当幽灵触碰到玩家时，玩家的生命值将会减一，同时，幽灵回到出生点。

```
else if (player_y-devil_y<20 && player_y-devil_y>0|| devil_y -player_y <20 &&devil_y -player_y >0)begin
                if (player_x-devil_x>0)begin
                    if (direction_reg==3)begin
                        if (cnt_count<10)begin
                            cnt_count<=cnt_count+2;
                        end
                        else begin
                            cnt_count<=0;
                        end
                        music<=3;
                    end
                    else begin
                        if (cnt_count<10)begin
                            cnt_count<=cnt_count+1;
                        end
                        else begin
                            cnt_count<=0;
                        end
                        music<=2;
                    end
                end
                else if (player_x-devil_x<0)begin
                    if (direction_reg==2)begin
                        if (cnt_count<10)begin
                            cnt_count<=cnt_count+2;
                        end
                        else begin
                            cnt_count<=0;
                        end
                        music<=3;
                    end
                    else begin
                       if (cnt_count<10)begin
                            cnt_count<=cnt_count+1;
                        end
                        else begin
                            cnt_count<=0;
                        end
                        music<=2;
                    end
                end
            end
if (direction_reg==0&&!(map[(devil_y+19)/20-1][devil_x/20]==1))begin
                    devil_y<=devil_y-1;
                end
                else if (direction_reg==1&&!(map[(devil_y/20)+1][devil_x/20]==1))begin
```

```
            devil_y<=devil_y+1;
        end
        else if (direction_reg==2&&!(map[(devil_y/20)][(devil_x+19)/20-1]==1))begin
            devil_x<=devil_x-1;
        end
        else if (direction_reg==3&&!(map[devil_y/20][(devil_x/20)+1]==1))begin
            devil_x<=devil_x+1;
        end
```

代码仿真如下

图 11 `ghost` 模块仿真

图 12 `ghost` 模块仿真(x 和 y 具体变化)

3.1.5 BCD 计数器

以下是部分代码展示

```
/*
该模块实现了一位 BCD 计数器功能，当 eat 信号为 1 时，功能启动。最终 pacman 模块采用了 4 位联级的 BCD 计数器
*/
always @(posedge Clk) begin
    if (Rst_n) begin
      cnt <= 4'd0;
    end
    else begin
      if(eat) begin
        if (Cin == 1'b1) begin
          if (cnt == 4'd9) begin
            cnt <= 4'd0;
          end
          else begin
            cnt <= cnt + 1'b1;
          end
        end
        else begin
          cnt <= cnt;
        end
      end
    end
```

```
    end
    assign Cout = (Rst_n==1)?0:(cnt==4'd9&&Cin==1'b1);
```

代码仿真如下

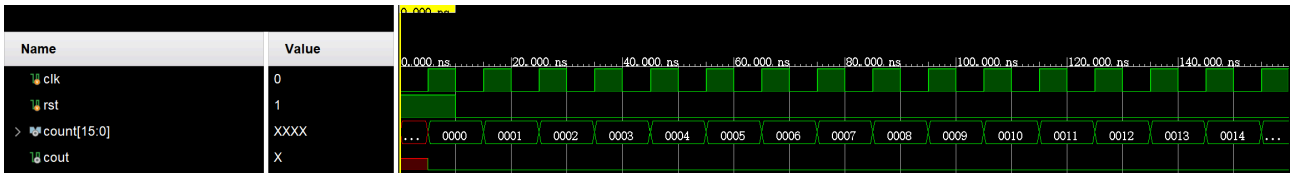

图 13　四位 BCD 模块仿真

3.2 VGA 屏幕显示

3.2.1 `vga_driver` 模块仿真时序分析

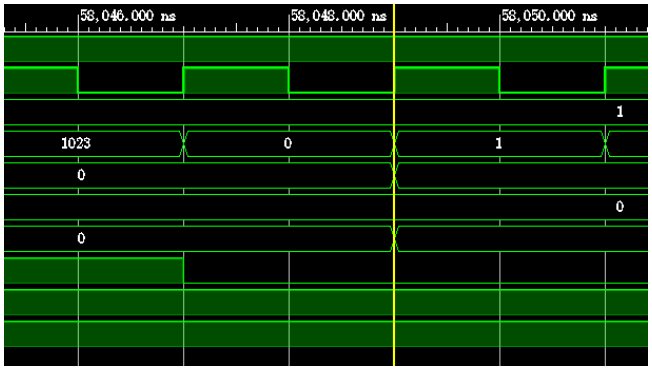

图 14　`vga_driver` 仿真波形局部

使用 `vga_driver_tb.v` 对 `vga_driver` 进行仿真，分析 `vga_driver` 的时序。如图 14 所示，`rdn` 会与 `col_addr` 同步提前一个 `vga_clk` 周期，这样就能通过 `rdn` 来判断 `col_addr`、`row_addr` 是否有效，并提前准备下一个周期需要显示的像素信息。

3.2.2 `renderer` 模块编写

`renderer` 模块需要根据 `row_addr` 和 `col_addr` 来提取相应位置的像素数据，也就是点对点渲染，这和电脑游戏逐帧渲染的思路有所不同。

对于每一个请求的像素位置 `row_addr` `col_addr`，判断这个像素是否在非背景区域内，如果是，则根据具体的情况调用相应的 ROM 获得像素信息；如果不是，则直接调取背景 ROM。由此，还产生了图层的概念，显然人物需要在最高图层，其次是 `dot`，地图之外由于没有元素重叠不需要考虑图层问题。

为了不产生时序问题，`renderer` 模块主要使用 `always @(*)` 组合电路，并使用 100MHz 的 `clk` 驱动 ROM 模块。这样，每个 `vga_clk` 周期内足够内存输入正确的地址，经过内部寄存器的缓冲，输出正确的数据。

```
//>>>>>>>>>>>>>>>>>>>> RENDERING <<<<<<<<<<<<<<<<<<<<//
always @(*) begin
    if(rst) begin
        rom_pacman_col_addr <= 0;
```

```
        rom_pacman_row_addr <= 0;
        rom_ghost_col_addr <= 0;
        rom_ghost_row_addr <= 0;
        rom_numbers_col_addr <= 0;
        rom_numbers_row_addr <= 0;
        rom_state_row_addr <= 0;
        rom_state_col_addr <= 0;
        rom_state_state <= 0;
        rom_dot_col_addr <= 0;
        rom_dot_row_addr <= 0;
        data_out <= 0;
    end else
    if(!rdn) begin
        //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> is ghost
        if(
            row_addr > 29 &&
            col_addr > 29 && // is in map
            row_addr - 30 > ghost_y - 1 &&
            row_addr - 30 < ghost_y + 20 &&
            col_addr - 30 > ghost_x - 1 &&
            col_addr - 30 < ghost_x + 20
        ) begin
            rom_ghost_row_addr <= (row_addr - 30) - ghost_y;
            rom_ghost_col_addr <= (col_addr - 30) - ghost_x;
            data_out <= rom_ghost_data;
        //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> is pacman
        end else if(
            row_addr > 29 &&
            col_addr > 29 && // is in map
            row_addr - 30 > pacman_y - 1 &&
            row_addr - 30 < pacman_y + 20 &&
            col_addr - 30 > pacman_x - 1 &&
            col_addr - 30 < pacman_x + 20
        ) begin
            rom_pacman_row_addr <= (row_addr - 30) - pacman_y;
            rom_pacman_col_addr <= (col_addr - 30) - pacman_x;
            data_out <= rom_pacman_data;
            // ......................
```

上面的代码展示了 `renderer` 主逻辑中的人物显示部分，判断之后得到了 `rom_row_addr` 和 `rom_col_addr`，这些内容将传给 `rom_controller`，然后在一个 `clk` 周期后，ROM ip 核输出了正确的数据，此时 `row_addr` 和 `col_addr` 仍然未变，数据输出 `data_out` 开始有效。

3.2.2.1 `rom_controller` 模块编写

内存地址 `rom_addr` 需要通过 `row_addr` 和 `col_addr` 换算得到，为了简化这一过程，我设计了 `rom_controller` 模块来封装 ROM ip 核，主要实现地址的转换，`rom_pacman_controller` 内容如下：

```
module rom_pacman_controller(
    input clk,
    input wire [1:0] direction, number,
    input wire [4:0] row_addr, col_addr,
    output wire [11:0] data
);

wire [12:0] rom_addr;
assign rom_addr = direction * 1600 + number * 400 + row_addr * 20 + col_addr;

rom_pacman_12_16_20_20 inst(
```

```
    .clka(clk),
    .addra(rom_addr),
    .douta(data)
);

endmodule
```

由于 pacman 有 4 种方向、每种方向 4 帧动画，所以每个方向造成 1600 的偏移，每帧造成 400 的偏移，每行造成 20 的偏移，这样就找到了正确的地址。

3.2.2.2 `cyclic_counter` 模块编写

pacman 的动画播放周期序列为`0, 1, 2, 3, 2, 1`，为了播放动画，我设计了`cyclic_4_conter`来周期性产生这个序列，并将结果输出到`pacman_rom_controller`的`number`。

```
`define interval 5 // 12 fps

module cyclic_4_counter( // 0 1 2 3 2 1,
    input v_sync, // frame by frame
    input rst,
    output wire [1:0] count
);

logic [1:0] array [0:5] = {
    2'b00, 2'b01, 2'b10, 2'b11, 2'b10, 2'b01
};

logic [3:0] interval_counter; // up to interval = 16
logic [2:0] count_value; // 0 to 5

assign count = array[count_value];

always @(negedge v_sync or posedge rst) begin
    if(rst) begin
        interval_counter <= 0;
        count_value <= 0;
    end else begin
        if(interval_counter == `interval - 1) begin
            interval_counter <= 0;
            count_value <= (count_value + 1) % 6;
        end else begin
            interval_counter <= interval_counter + 1;
        end
    end
end
```

而 ghost 动画只设计了 2 帧，就不需要这么复杂了。

3.2.3 VGA 静态显示测试

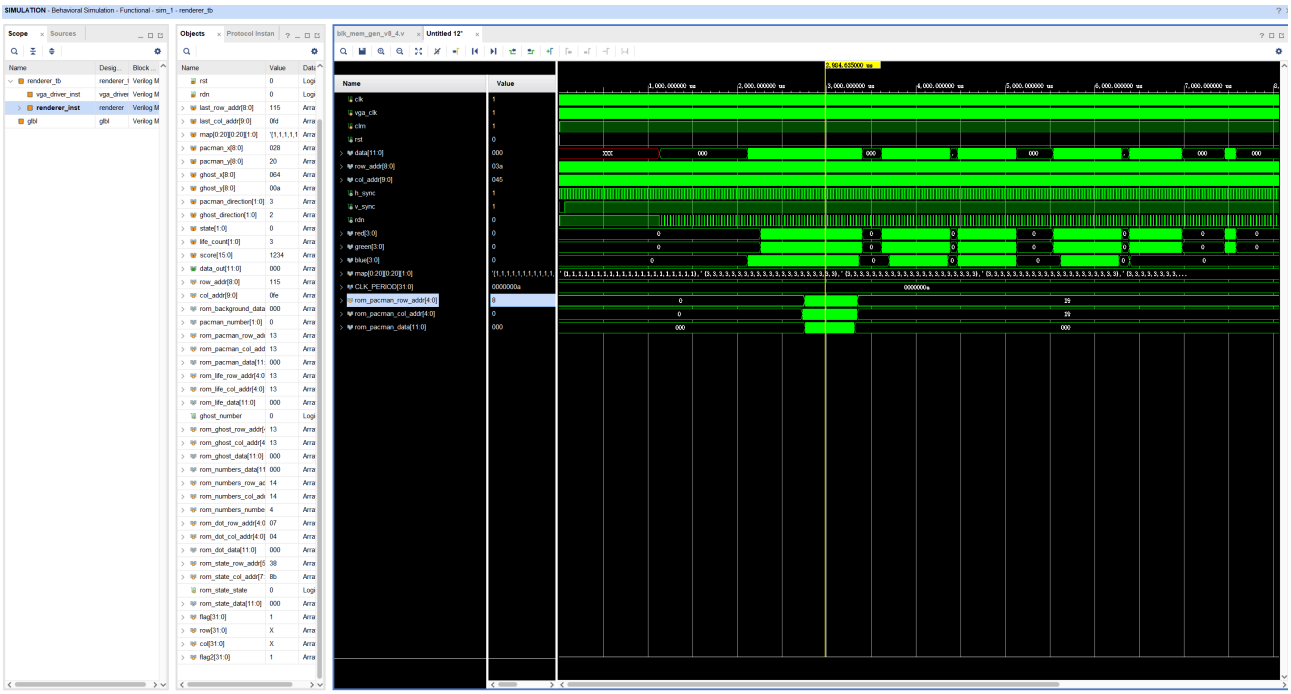

图 15 `display` 模块仿真部分波形

由于 Vivado 仿真 `display` 模块显示逐帧动画的速度比较慢，且波形较长（如图 15，最长的动画需要仿真持续 1s 的波形）会引起卡顿，我只对 `display` 模块进行了一次完整仿真，而更多使用静态显示测试来验证模块的功能。静态仿真中已经发现 `renderer` 模块能够在一个 `vga_clk` 周期内及时相应 `vga_driver` 模块对像素信息的请求，每个 `posedge vga_clk` 对应的像素信息都是正确的。

静态显示测试将人物固定在确定的位置，给定特殊的地图，显示特定的分数和状态，观察显示是否正常，人物动画是否正常播放。

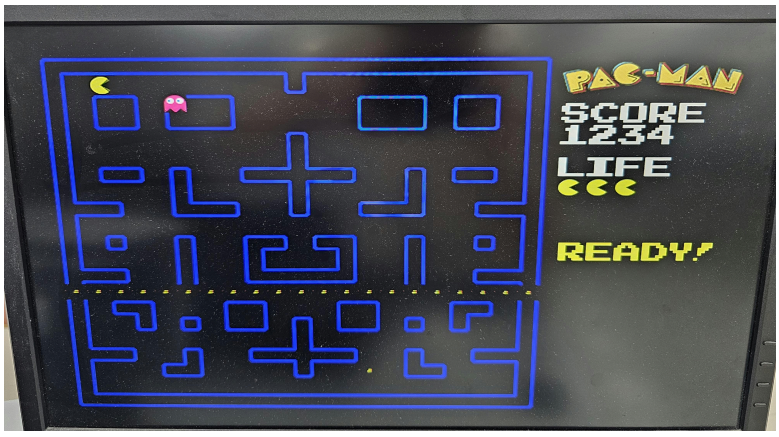

图 16 `display` 模块静态测试照片

从图 16 可以看出，VGA 显示时序没有出现任何问题，所有像素点都正常显示。而且分数、剩余生命、游戏状态、人物动画都正常显示。另外，注意到人物的图层在背景之上，dot 的图层也在背景之上。测试的结果符合预期。

3.3 蜂鸣器音乐播放

3.3.1 `buzzer_driver` 模块编写

`buzzer_driver` 已经内已经存储了音高对应的频率，下面实现针对某个频率的时钟分频 `beep` 驱动。

```
always @(posedge clk or posedge rst) begin
    if(rst) begin
        counter <= 0;
        beep <= 0;
    end else begin
        if(note < 48) begin // valid note
            threshold <= CLK_FREQ / (2 * note_freq[note]);
            // counter imp
            if(counter >= threshold) begin
                counter <= 0;
                beep <= ~beep;
            end else begin
                counter <= counter + 1;
            end
        end else begin // invalid note, or stop
            threshold <= 0; // stop buzzer
            beep <= 0;
        end
    end
end
```

上面这段代码实现了 reset 功能。在每个时钟周期记录当前音符编码输入对应的音高半周期 `threshold`，并实现一个计数器，每到 `threshold` 进行一次 `beep` 信号的反转，从而输出当前播放音高对应的方波信号来驱动蜂鸣器。

3.3.2 `buzzer_sync` 模块编写

`buzzer_sync` 需要在内部根据 `music_selection` 选择正确的内置音乐音效来播放。注意吞食豆子音效并非 `music_selection` 选择，而是使用 `eat` 信号的短暂脉冲激活，吞食豆子音效的播放有最高优先级，需要盖过正常游戏的背景音乐。下面是 `buzzer_sync` 模块的主要逻辑部分。

```
always @(posedge clk_ms or posedge rst) begin
    // reset
    if(rst) begin
        music_state <= `IDLE; // set idle
        counter <= 0;
        eat_state <= 0; // not eating
        note <= `empty; // no sound
        eat_state_reg <= 0;
    // other
    end else if(eat_fruit) begin
        eat_state <= 1;
    end else begin
        // eat
        if(eat_state) begin // Eating state
            if(eat_counter < eat_time) begin
                note <= eat[eat_counter / eat_note_time]; // play the note for this ms
                eat_counter <= eat_counter + 1;
            end else begin
                eat_state <= 0; // stop eating
```

```
                eat_counter <= 0;
            end
        // else
        end else begin // Music state
            if(music_state != music_selection) begin
                music_state <= music_selection;
                counter <= 0;
            end else begin
                counter <= counter + 1;
                case(music_state)
                    `IDLE: note <= `empty; // idle, no sound
                    `STARTUP: begin
                        if(counter < startup_time) begin
                            note <= startup[counter / startup_note_time]; // play the note for this ms
                        end else begin
                            music_state <= `IDLE; // set idle
                            counter <= 0; // reset counter
                        end
                    end
                    `GAMEOVER: begin
                        if(counter < gameover_time) begin
                            note <= gameover[counter / gameover_note_time]; // play the note for this ms
                        end else begin
                            music_state <= `IDLE; // set idle
                            counter <= 0;
                        end
                    end
                    `SLOW: begin
                        if(counter >= slow_time) counter <= 0;
                        note <= slow_and_fast[counter / slow_note_time]; // play the note for this ms
                    end
                    `FAST: begin
                        if(counter >= fast_time) counter <= 0;
                        note <= slow_and_fast[counter / fast_note_time]; // play the note for this ms
                    end
                    default: begin
                        music_state <= `IDLE; // set idle if invalid input
                        counter <= 0;
                    end
                endcase
            end
        end
    end
end
```

3.3.3 `buzzer_sync` 模块仿真

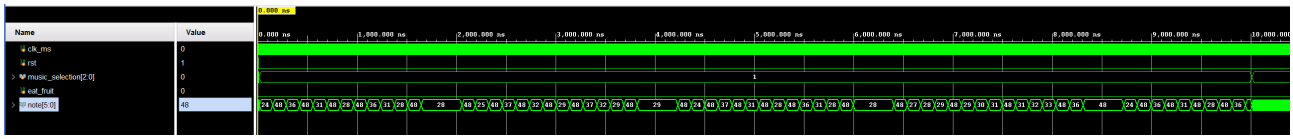

图 17 `buzzer_sync` 仿真波形：正常播放 Startup 音乐

由于 `music` 模块的整体仍然存在仿真时间过长的问题，此处仅对 `buzzer_sync` 进行仿真，且输入的 `clk_ms` 实际上是由 ns 时钟驱动的。根据图 17 可以看出，`buzzer_sync` 正确输出了图 7 中设计的 Startup 音乐。

3.3.4 `music` 模块测试

将 `SW[1:0]` 映射到 `music_selection`，将 `SW[2]` 映射到 `eat_fruit` 并在板上进行测试。

音乐测试无法通过照片展示。测试发现`music`模块功能正常，会循环播放`music_selection`指定的音乐。我们发现，实际上在`eat_fruit`的负边沿才会播放吃豆音效，但是考虑到游戏模块给的脉冲持续时间不会超过 1ms，实际游玩时没有任何影响。

3.4 PS/2 键盘输入

以下是部分代码展示，这是关于最终输出的判定

```
always @(posedge clk) begin
    pre_up=up;
    pre_down=down;
    pre_left=left;
    pre_right=right;
  case (data)
        10'h272:down= 1;
        10'h372:down= 0;
        10'h29:space= 1;
        10'h129:space= 0;
        10'h275:up= 1;
        10'h375:up= 0;
        10'h26B:left= 1;
        10'h36B:left= 0;
        10'h274:right = 1;
        10'h374:right= 0;
    endcase
    if(pre_up==0&&up==1)begin
        direction=0;
    end
    if(pre_down==0&&down==1)begin
        direction=1;
    end
    if(pre_left==0&&left==1)begin
        direction=2;
    end
    if (pre_right==0&&right==1)begin
        direction=3;
    end
end
```

为了保证代码的正确性，我将 direction 输出连接到 LED 上，根据不同的按键来决定 LED 的亮灭，以下是下板照片

图 18 `ps2`下板照片

4 整体测试与分析

4.1 功能展示

游戏开始时，开始音乐响起，豆人根据键盘输入方向开始移动，鬼魂出现并开始移动游戏音乐变得缓和，吃豆人每吃一个豆子计分加 1，蜂鸣器产生吃豆子的音效

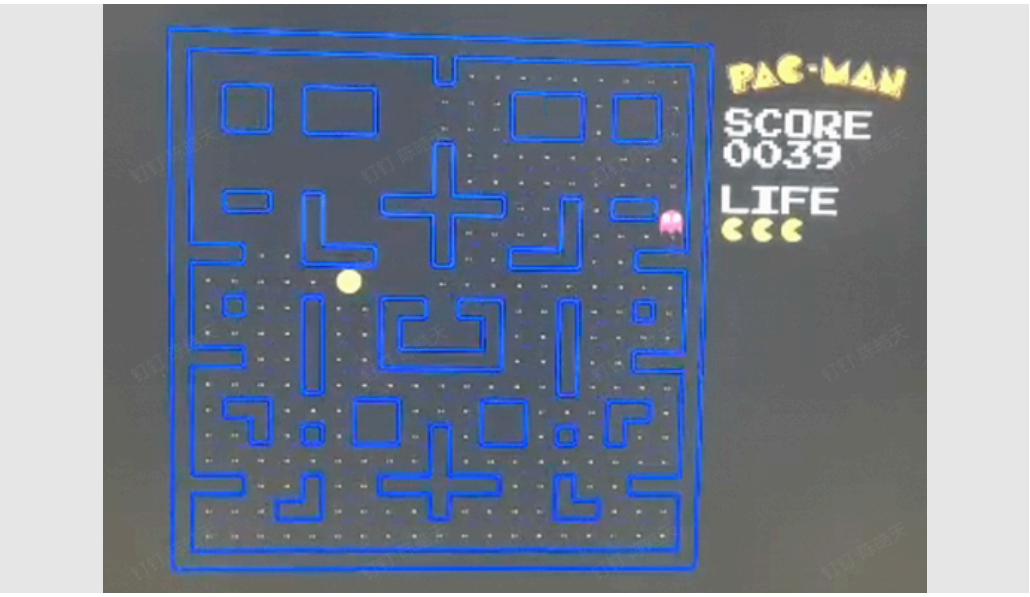

图 19　游戏正常运行照片

鬼魂碰到墙之后随机选择方向并前进，如果吃豆人在鬼的前进方向上，音乐将变得急促。如果鬼魂与吃豆人进行了碰撞，吃豆人生命值将会减一，鬼魂回到出生点。

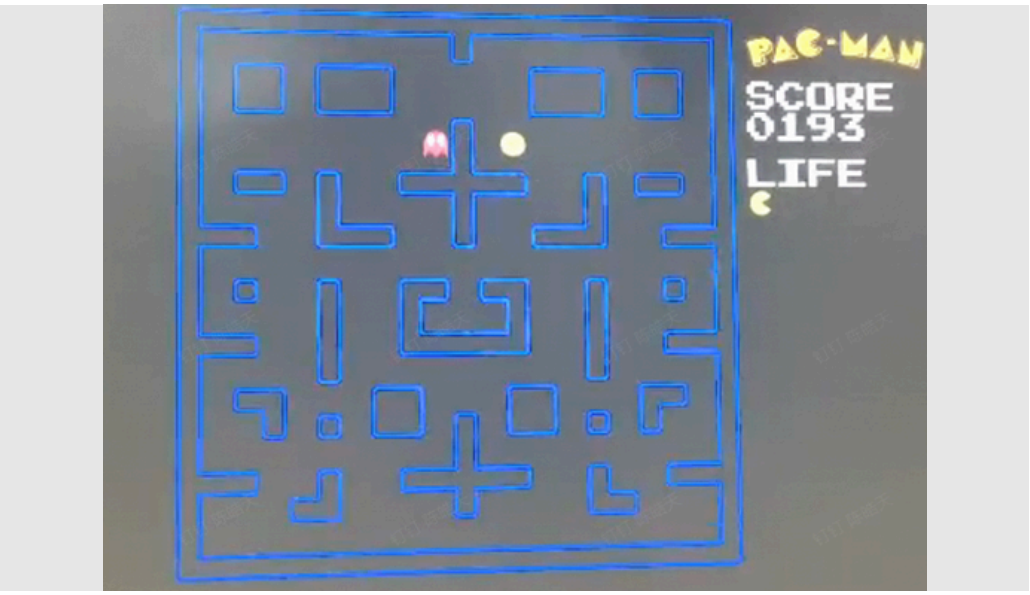

图 20　生命值减少的照片

当吃豆人生命值归零时，游戏播放结束音乐，两者回到起始点。

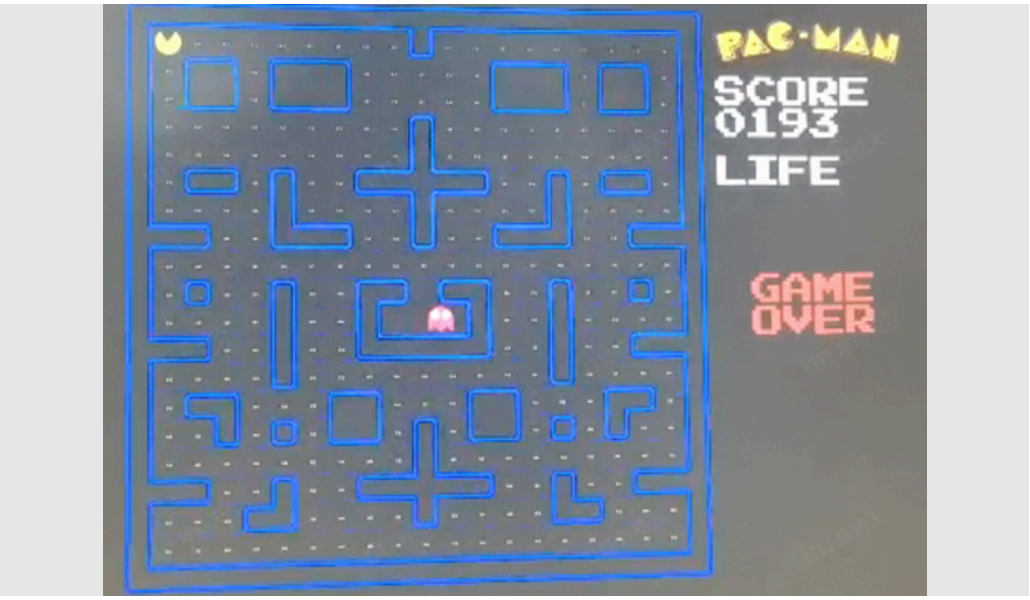

图 21　生命值减少的照片

4.2 分析

当前模块仍然存在一些问题，一是吃豆人由于碰撞面积较大，进入狭小的通道可能会有困难，需要贴着墙壁进入或者反复手动调位。

二是幽灵的方向“随机”选择严重依赖于吃豆人的分数大小，当吃豆人不再获取分数时，幽灵可能会在很长一段时间内沿相同的路线走。

其三，游戏的平衡性有待提升，为了保证游戏的平衡性，我们降低了玩家的移动速度，但这也导致了吃完豆子所需要的时间过长，面对鬼魂的追击无力躲避等等问题，这些问题可以被进一步解决。

5 结论与展望

5.1 遇到的问题和解决过程

5.1.1 Clock Wizard ip 核引起的时序违例问题

VGA 显示中的驱动时钟`vga_clk`原本计划使用 Clock Wizard ip 核产生标准的 25.175MHz 时钟信号。使用这个 ip 核，我跑通了小节 3.2.3 的静态显示测试，但是在后面构建整个工程的过程中，我发现 Vivado2022 会卡在 implementation 的 route_design 部分一个多小时也没有反应。

善用搜索后，我研究了 implementation 的 log，发现 Vivado 在布线优化中总存在 WNS(Worst Negative Slack)为负数，即无法满足时序要求的问题，故而会卡在线路时序优化算法中无法退出。但是由于时序报告只会在 implementation 结束后才生成，我无法定位产生时序违例的线路。

所以，我尝试使用 Vivado2024.1 来构建工程，终于跑通了 implementation，在时序报告中发现了如图 22 所示的时序违例线路。

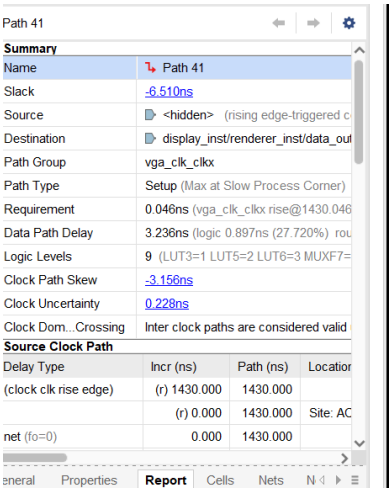

图 22　Vivado2024 生成的时序报告中对违例线路的描述

图 22 报告指出，从`<hidden>`（实际上就是 ROM ip）到`renderer`的线路有时序违例情况，但是这并不是因为逻辑过于复杂，而是时序要求`requirement`仅有 0.046ns。这是因为像素数据从 ROM ip 核读取，ROM ip 核由`clk`驱动；而像素数据在`vga_driver`中被使用，`vga_driver`由`vga_clk`驱动；由于两个时钟的频率没有整数倍关系，两个时钟上升沿间的最短间隔仅有 0.046ns，这就产生了离谱的时序要求。

从这里发我们已经能够得出结论，ROM ip 的时钟和`vga_clk`的频率一定需要成整数倍的关系。为此，我们弃用了 Clock Wizard ip 核，使用`clk_div`来得到 25MHz 的`vga_clk`。

但是，这样会导致实验室的显示器出现兼容性问题，显示器无法识别 25MHz 这个不太准确的频率。我想到在约束文件中可以自定义系统时钟的频率，按照 $25.175 \times 4 = 100.7$ 计算，我们需要的系统时钟频率为 100.7MHz，对应的时钟周期约为 9.9305ns。

修改之后，屏幕正常显示。

我认为我们对于这一问题的修正方法具有指导和示范意义。我们建议不要使用 Clock Wizard ip 核来满足`vga_clk`的频率要求，并且尽量使用最新版的 Vivado 来完成课程设计。另外，我们认为存在其他更好的解决方法，关键也在于保证内存驱动时钟的频率是`vga_clk`的正整数倍。

5.1.2 VGA 显示噪点问题

在静态测试中，VGA 显示没有出现任何噪点，动画播放流畅。但是，在早期的成果上板中，我发现人物仍然存在闪烁问题。

我推测这一问题是由于`game_interface`模块传来的人物方向和 xy 坐标一直在抖动，导致连续多帧出现内存寻址错误。我怀疑这是因为碰撞判断中方向和坐标输出寄存器的内容被反复修改，我们讨论后修改了碰撞判断的实现，最终人物动画正常播放，没有任何噪点。

5.1.3 吃豆人模块实现中遇到的问题

5.1.3.1 数组调用

在构建地图时，我们使用到了数组。但是我们发现，verilog 语言是不支持数组调用的。于是在查询资料后，我们发现了 system verilog 语言的强大。它不仅能调用数组，还拥有 logic 变量声明，可以同时拥有 reg 和 wire 的功能。

5.1.3.2 边缘检测

在判断吃豆人是否会撞墙或者吃豆时，我们发现不是很容易，不能简单地将吃豆人坐标抽象成左上角坐标值来进行“离散”的判断，而是要使用碰撞箱进行模拟

5.1.3.3 multidriven 问题（vivado 报错）

在进行综合和布线时，刚开始我发现 vivado 报了 multidriven 的问题，经过研究，发现是在多个不同 always 块中改变了同一个值或是在不同模块改变了相同的输入/输出。

5.2 讨论、心得

5.2.1 by 陈皓天

通过本次作业，我对时序电路的理解更为加深了。特别是在实现鬼魂的移动模块和 PS2 模块的过程中，我学到了如何处理时序信号、设计状态机以及优化时序逻辑。这让我更加熟练地应用课堂所学的理论知识，也使我对数字系统设计有了更全面的认识。希望今后能够进一步挑战更复杂的时序电路设计，不断提升自己在数字电路方面的能力。

5.2.2 by 程韬

我在这次课程设计中担任组长。我发现这次组内合作工作流中仍然存在很多问题。首先是版本控制问题，一方面我们小组整体上对 git 工具并不熟悉，所以采用了比较原始的钉钉群文件方式来进行进度同步；另一方面，在成果下板调试时，没有及时备份每一次修改产生的比特流和源代码，这也造成了不小的麻烦。在撰写报告时，我们吸取经验教训，使用 typst 小组功能线上共同撰写并排版，大大提高了合作效率。

这次作业中我专攻输出交互方面。我找了很多资料，耗尽了自己的不多的乐理知识，才完成了音乐音效的设计；其中最令我满意的是 Startup 音乐，得益于蜂鸣器特有的方波音色，我们几乎完美复刻了 Pacman 标志性的开场音乐，致敬了这一款经典游戏。我熟练使用 Photoshop 绘制素材，设计逐帧动画，做到了 ghost 眼睛看向前进方向这个细节，这也让我很有成就感。

此外，我还在一次次解决 bug 的过程中对 Vivado 产生了更深刻的了解。我从 Xilinx 用户论坛上学到了如何看时序报告、如何解决 log 不输出到控制台的问题、以及如何通过 schmetic 功能来分析电路问题。

5.2.3 by 靳科源

通过本次大作业，我对于 verilog 语言有了一个更加深刻的理解，对组合逻辑电路与时序电路的理解加深了。在实现吃豆人移动与地图构建时，我学会如何利用 BCD 计数器进行技术、时钟分频、“碰撞箱”判断是否撞墙等等，有很多收获。希望以后能继续学习实践数字系统设计，对计算机硬件加深认识。

5.3 展望

本次作业虽然实现的不错，但是仍然存在很多方面可以进一步提升。

首先是游戏逻辑。由于种种困难，最终实现的游戏逻辑并没有完成一开始讨论出的设计方案。例如，游戏中没有实现预操作功能，而且碰撞箱宽度刚好等于最窄过道宽度，这就导致吃豆人很难通过直接转弯进入一格的通道。

其次是显示效果。VGA 显示的最终效果过于简单，只有一张背景。如果游戏更加复杂，可以尝试制作游戏菜单页面、游戏结算页面以及转场动画等等。这些实现只需要绘制更多素材，分配更多状态即可完成。

6 附录

6.1 小组主要工作说明

我们参考的工程如下：

- 优秀报告《FPGA 纸上小鸟游戏》：学习渲染模块编写思路
- 优秀报告《基于数字系统的迷宫》：学习报告的格式和内容要求

我们直接使用的代码如下：

- 实验文档 https://guahao31.github.io/2024_DD/final_project/device/ 中的 `vgac` 模块
- 实验文档 https://guahao31.github.io/2024_DD/lab7/multiplexer/#displaynumber 中的 `clkdiv` 模块

我们参考的其他在线资源如下：

- https://freepacman.org/ 吃豆人在线游戏：参考游戏逻辑、音乐和图像
- https://onlinesequencer.net/ 在线编曲网站：简谱搜索和音乐设计
- Xilinx 论坛资源
  - https://support.xilinx.com/s/question/0D52E00006hpo9XSAQ/vivado%E4%B8%80%E7%9B%B4%E5%81%9C%E5%9C%A8-running-routedesign?language=en_US
    解决 Vivado 停在 Running route_design 问题
  - https://support.xilinx.com/s/question/0D52E00006hpogmSAA/vivado%E6%B2%A1%E6%9C%89log%E8%BE%93%E5%87%BA?language=en_US
    解决控制台不输出 log 问题
  - https://support.xilinx.com/s/question/0D52E00006hpdTfSAI/error-common-171284-badly-structured-ieee1735-one-of-the-encrypted-files-could-not-be-decrypted?language=en_US
    解决创建 ip 核时的“加密文件无法被解密“报错

除了上述参考资料之外，我们的其他工作均由组内成员合作完成。在此基础上，我们进行的主要工作总结如下：

- 学习与研究
  - 研究吃豆人游戏的逻辑和规则
  - 学习 System Verilog 语言语法
- 设计与开发
  - 顶层模块设计和系统集成
  - 游戏逻辑设计
  - 外设接口设计
- 编码与实现
  - 编写 System Verilog 代码实现所有游戏功能
  - 编写 System Verilog 代码实现外设输入输出
  - 设计和处理游戏素材
- 测试与调试
  - 进行模块级仿真测试

  - 进行系统级仿真和调试
  - 进行下板实现和调试
- 成果展示
  - 拍摄、剪辑展示视频
  - 撰写课程设计报告

小组的具体工作请参考报告前文及贡献度表。

6.2 组员分工及贡献度表

姓名	学号	主动述职	贡献度	签名
cht		鬼魂模块编写 ps2 模块编写 报告撰写、视频剪辑		
Chen Tao		组长 VGA 显示实现和素材绘制 蜂鸣器播放预置音乐实现 报告撰写		
lao jin		吃豆人与游戏功能总模块编写 地图构建 报告撰写 构建 BCD 计数器统计分数		

6.3 工程代码

6.3.1 `ps2.v`

```
always @(posedge clk) begin
    pre_up=up;
    pre_down=down;
    pre_left=left;
    pre_right=right;
  case (data)
        10'h272:down= 1;
        10'h372:down= 0;
        10'h29:space= 1;
        10'h129:space= 0;
        10'h275:up= 1;
        10'h375:up= 0;
        10'h26B:left= 1;
        10'h36B:left= 0;
        10'h274:right = 1;
        10'h374:right= 0;
    endcase
    if(pre_up==0&&up==1)begin
        direction=0;
    end
    if(pre_down==0&&down==1)begin
        direction=1;
    end
    if(pre_left==0&&left==1)begin
```

```
        direction=2;
    end
    if (pre_right==0&&right==1)begin
        direction=3;
    end
end
```

6.3.2 `map.v`

```
always @(posedge reset) begin
        // if(reset) begin
            for(i=0;i<21;i=i+1) begin
                for(j=0;j<21;j=j+1) begin
                    map[i][j] <= 2'b11;
                end
            end
            for(i=0;i<21;i=i+1) begin
                map[0][i] <= 1;
                map[20][i] <= 1;
            end

            for(i=0;i<21;i=i+1) begin
                map[i][0]<=1;map[i][20]<=1;
            end

            map[1][1]<=0; //initial location of pac man
            //2 5 6 17
            map[1][10]<=1;map[4][10]<=1;map[5][10]<=1;map[16][10]<=1;
            //3
            map[2][2]<=1;map[2][3]<=1;map[2][5]<=1;map[2][6]<=1;map[2][7]<=1;
            map[2][13]<=1;map[2][14]<=1;map[2][15]<=1;map[2][17]<=1;map[2][18]<=1;
            //4
            map[3][2]<=1;map[3][3]<=1;map[3][5]<=1;map[3][6]<=1;map[3][7]<=1;
            map[3][13]<=1;map[3][14]<=1;map[3][15]<=1;map[3][17]<=1;map[3][18]<=1;
            //7
            map[6][2]<=1;map[6][3]<=1;map[6][5]<=1;map[6][8]<=1;map[6][9]<=1;map[6][10]<=1;
            map[6][18]<=1;map[6][17]<=1;map[6][15]<=1;map[6][12]<=1;map[6][11]<=1;
            //8
            map[7][5]<=1;map[7][10]<=1;map[7][15]<=1;
            //9
            map[8][2]<=1;map[8][1]<=1;map[8][5]<=1;map[8][6]<=1;map[8][7]<=1;map[8][10]<=1;
            map[8][19]<=1;map[8][1]<=1;map[8][15]<=1;map[8][13]<=1;map[8][14]<=1;
            //11
            map[10][2]<=1;map[10][5]<=1;map[10][8]<=1;map[10][9]<=1;
            map[10][10]<=0;
            map[10][18]<=1;map[10][15]<=1;map[10][12]<=1;map[10][11]<=1;
            //12
            map[11][5]<=1;map[11][8]<=1;map[11][15]<=1;map[11][12]<=1;
            //13
            map[12][1]<=1;map[12][2]<=1;map[12][5]<=1;map[12][8]<=1;map[12][9]<=1;map[12][10]<=1;
            map[12][19]<=1;map[12][18]<=1;map[12][15]<=1;map[12][12]<=1;map[12][11]<=1;
            //14
            map[13][5]<=1;map[13][15]<=1;
            //15
            map[14][2]<=1;map[14][3]<=1;map[14][8]<=1;map[14][7]<=1;
            map[14][18]<=1;map[14][17]<=1;map[14][12]<=1;map[14][13]<=1;
            //16
            map[15][3]<=1;map[15][5]<=1;map[15][7]<=1;map[15][8]<=1;map[15][10]<=1;
            map[15][17]<=1;map[15][15]<=1;map[15][13]<=1;map[15][12]<=1;
            //18
            map[17][1]<=1;map[17][2]<=1;map[17][5]<=1;map[17][8]<=1;map[17][9]<=1;map[17][10]<=1;
```

```
            map[17][19]<=1;map[17][18]<=1;map[17][15]<=1;map[17][12]<=1;map[17][11]<=1;
            //19
            map[18][4]<=1;map[18][5]<=1;map[18][10]<=1;
            map[18][16]<=1;map[18][15]<=1;
//        end
    end
```

6.3.3 `sys_rst`

```
module sys_rst(
    input sw,
    input clk,
    output reg rst
);

logic [3:0] power_on_counter = 4'd0;
logic power_on_rst = 1'b1;

always @(posedge clk) begin
    // power on reset
    if (power_on_rst) begin
        if (power_on_counter < 4'd15) begin
            power_on_counter <= power_on_counter + 1'b1;
            rst <= 1'b1;
        end else begin
            power_on_rst <= 1'b0;
            rst <= sw;
        end
    end else begin
        // controlled by switch
        rst <= sw;
    end
end

endmodule
```

6.3.4 `music`

```
module music(
    input rst, clk,
    input [2:0] music_select,
    input eat,
    output beep
);

wire clk_ms;
wire [5:0] note;

clk_ms_1 clk_ms_inst(
    .clk(clk),
    .clk_ms(clk_ms)
);

buzzer_sync buzzer_sync_inst(
    .clk_ms(clk_ms),
    .rst(rst),
    .eat_fruit(eat),
    .music_selection(music_select),
```

```
    .note(note)
);

buzzer_driver buzzer_driver_inst(
    .clk(clk),
    .rst(rst),
    .note(note),
    .beep(beep)
);

endmodule
```

6.3.4.1 `buzzer_driver`

```
module buzzer_driver(
    input logic clk,
    input logic [5:0] note, // ref to note_freq[]
    input logic rst,
    output logic beep
);

// register and param
logic [31:0] counter;
integer threshold;

parameter CLK_FREQ = 100_000_000;

integer note_freq [0:47] = {
  // C,   C#,  D,   D#,  E,   F,   F#,  G,   G#,  A,   A#,  B
    131, 139, 147, 156, 165, 175, 185, 196, 208, 220, 233, 247, // C3 - B3
    262, 277, 294, 311, 330, 349, 370, 392, 415, 440, 466, 494, // C4 - B4
    523, 554, 587, 622, 659, 698, 740, 784, 831, 880, 932, 988, // C5 - B5
    1047, 1109, 1175, 1245, 1319, 1397, 1480, 1568, 1661, 1760, 1865, 1976  // C6 - B6
};


always @(posedge clk or posedge rst) begin
    if(rst) begin
        counter <= 0;
        beep <= 0;
    end else begin
        if(note < 48) begin // valid note
            threshold <= CLK_FREQ / (2 * note_freq[note]);
            // counter imp
            if(counter >= threshold) begin
                counter <= 0;
                beep <= ~beep;
            end else begin
                counter <= counter + 1;
            end
        end else begin // invalid note, or stop
            threshold <= 0; // stop buzzer
            beep <= 0;
        end
    end
end

endmodule
```

6.3.4.2 `buzzer_sync`

```
// define the notes and their index
`define C3 0
`define C_3 1
`define D3 2
`define D_3 3
`define E3 4
`define F3 5
`define F_3 6
`define G3 7
`define G_3 8
`define A3 9
`define A_3 10
`define B3 11
`define C4 12
`define C_4 13
`define D4 14
`define D_4 15
`define E4 16
`define F4 17
`define F_4 18
`define G4 19
`define G_4 20
`define A4 21
`define A_4 22
`define B4 23
`define C5 24
`define C_5 25
`define D5 26
`define D_5 27
`define E5 28
`define F5 29
`define F_5 30
`define G5 31
`define G_5 32
`define A5 33
`define A_5 34
`define B5 35
`define C6 36
`define C_6 37
`define D6 38
`define D_6 39
`define E6 40
`define F6 41
`define F_6 42
`define G6 43
`define G_6 44
`define A6 45
`define A_6 46
`define B6 47
`define empty 48 // rest note

/***********
* About music selection:
* 0: idle
* 1: startup music
* 2: slow music
* 3: fast music
* 4: gameover music
*/

`define IDLE 3'b000
`define STARTUP 3'b001
`define SLOW 3'b010
`define FAST 3'b011
```

```
`define GAMEOVER 3'b100

module buzzer_sync(
    input logic [2:0] music_selection,
    input logic eat_fruit,
    input logic rst,
    input logic clk_ms,
    output logic [5:0] note
);

// start up music
integer startup_time = 4352; // ms
integer startup_note_time = 68; // ms
integer startup [0:63] = {
    `C5, `empty,  `C6, `empty, `G5, `empty, `E5, `empty, `C6, `G5, `E5, `empty, `E5, `E5, `E5, `empty,
    `C_5, `empty, `C_6, `empty, `G_5, `empty, `F5, `empty, `C_6, `G_5, `F5, `empty, `F5, `F5, `F5,
`empty,
    `C5, `empty, `C_6, `empty, `G5, `empty, `E5, `empty, `C6, `G5, `E5, `empty, `E5, `E5, `E5, `empty,
    `D_5, `E5, `F5, `empty, `F5, `F_5, `G5, `empty, `G5, `G_5, `A5, `empty, `C6, `empty, `empty, `empty
};

// gameover music
integer gameover_time = 2176; // ms
integer gameover_note_time = 136; // ms
integer gameover [0:15] = {
    `D6, `empty, `C_6, `empty, `C6, `empty, `B5, `empty, `A_5, `empty, `D_3, `empty, `D_3, `empty,
`empty, `empty
};

// slow music
integer slow_time = 374; // ms
integer slow_note_time = 17; // ms
integer fast_time = 220; // ms
integer fast_note_time = 10; // ms
integer slow_and_fast [0:21] = {
    `B4, `C5, `C_5, `D5, `D_5, `E5, `F5, `F_5, `G5, `G_5, `A5,
    `A_5, `A5, `G_5, `G5, `F_5, `F5, `E5, `D_5, `D5, `C_5, `C5
};

// eat music
integer eat_time = 200; // ms
integer eat_note_time = 25; // ms
integer eat [0:7] = {
    `C_5, `F_5, `A5, `empty, `B5, `F5, `A_4, `empty
};

logic [2:0] music_state;
integer counter;
logic eat_state;
logic eat_state_reg;
integer eat_counter;

always @(posedge clk_ms or posedge rst) begin
    // reset
    if(rst) begin
        music_state <= `IDLE; // set idle
        counter <= 0;
        eat_state <= 0; // not eating
        note <= `empty; // no sound
        eat_state_reg <= 0;
    // other
    end else if(eat_fruit) begin
        eat_state <= 1;
    end else begin
```

```
        // eat
        if(eat_state) begin // Eating state
            if(eat_counter < eat_time) begin
                note <= eat[eat_counter / eat_note_time]; // play the note for this ms
                eat_counter <= eat_counter + 1;
            end else begin
                eat_state <= 0; // stop eating
                eat_counter <= 0;
            end
        // else
        end else begin // Music state
            if(music_state != music_selection) begin
                music_state <= music_selection;
                counter <= 0;
            end else begin
                counter <= counter + 1;
                case(music_state)
                    `IDLE: note <= `empty; // idle, no sound
                    `STARTUP: begin
                        if(counter < startup_time) begin
                            note <= startup[counter / startup_note_time]; // play the note for this ms
                        end else begin
                            music_state <= `IDLE; // set idle
                            counter <= 0; // reset counter
                        end
                    end
                    `GAMEOVER: begin
                        if(counter < gameover_time) begin
                            note <= gameover[counter / gameover_note_time]; // play the note for this ms
                        end else begin
                            music_state <= `IDLE; // set idle
                            counter <= 0;
                        end
                    end
                    `SLOW: begin
                        if(counter >= slow_time) counter <= 0;
                        note <= slow_and_fast[counter / slow_note_time]; // play the note for this ms
                    end
                    `FAST: begin
                        if(counter >= fast_time) counter <= 0;
                        note <= slow_and_fast[counter / fast_note_time]; // play the note for this ms
                    end
                    default: begin
                        music_state <= `IDLE; // set idle if invalid input
                        counter <= 0;
                    end
                endcase
            end
        end
    end
end

endmodule
```

6.3.4.3 clk_ms

```
module clk_ms_1(
  input logic clk,
 output logic clk_ms
);
```

```
  logic [31:0] cnt;

 initial begin
    cnt <= 32'b0;
    clk_ms <= 0;
 end

 wire [31:0] cnt_next;
  assign cnt_next = cnt + 1'b1;

 always @(posedge clk) begin
    if(cnt<50_000)begin
      cnt <= cnt_next;
   end
    else begin
     cnt <= 0;
      clk_ms <= ~clk_ms;
   end
  end

endmodule
```

6.3.5 `display`

```
module display(
    //>>>>> input <<<<<//
    input rst, clk,
    input wire [1:0] map [0:20] [0:20],
    input wire [8:0] pacman_x, pacman_y, ghost_x, ghost_y,
    input wire [1:0] pacman_direction, ghost_direction, state, life_count,
    input wire [15:0] score,
    //>>>>> output <<<<<//
    output wire [3:0] vga_red, vga_green, vga_blue,
    output wire vga_hsync, vga_vsync
);

wire clrn, vga_clk, rdn;
wire [11:0] data;
wire [8:0] row_addr;
wire [9:0] col_addr;
assign clrn = ~rst; // active low

//clkx clkx_inst(
//    .clk(clk),
//    .vga_clk(vga_clk),
//    .reset(rst)
//);

wire [31:0] div_res;
assign vga_clk = div_res[1];

clkdiv clkdiv_inst(
    .clk(clk), .rst(rst), .div_res(div_res)
);

vga_driver vga_driver_inst(
    .vga_clk(vga_clk),
    .d_in(data),
    .clrn(clrn),
    .row_addr(row_addr),
    .col_addr(col_addr),
```

```
    .r(vga_red),
    .g(vga_green),
    .b(vga_blue),
    .rdn(rdn),
    .hs(vga_hsync),
    .vs(vga_vsync)
);

renderer renderer_inst(
    //>>>>> input <<<<<//
    .v_sync(vga_vsync),
    .clk(clk),
    .rst(rst),
    .rdn(rdn),
    .row_addr(row_addr),
    .col_addr(col_addr),
    .map(map),
    .pacman_x(pacman_x),
    .pacman_y(pacman_y),
    .ghost_x(ghost_x),
    .ghost_y(ghost_y),
    .pacman_direction(pacman_direction),
    .ghost_direction(ghost_direction),
    .state(state),
    .life_count(life_count),
    .score(score),
    //>>>>> output <<<<<//
    .data_out(data)
);

endmodule
```

6.3.5.1 `renderer`

```
// for every grid in input minimap
`define grid_have_dot 2'b11
`define grid_is_empty 2'b00
`define grid_have_wall 2'b01
`define state_ready 2'b00
`define state_gameover 2'b11
`define state_idle 2'b01

module renderer(
    // clock
    input v_sync, // v_sync is used to track the frame
    input clk, rst,
    // from vga_driver
    input rdn,
    input wire [8:0] row_addr,
    input wire [9:0] col_addr,
    // from game
    input wire [1:0] map [0:20] [0:20],
    input wire [8:0] pacman_x, pacman_y, ghost_x, ghost_y,
    input wire [1:0] pacman_direction, ghost_direction,
    input wire [1:0] state,
    input wire [1:0] life_count,
    input wire [15:0] score, // BCD 4 digit game score
    // to vga_driver
    output logic [11:0] data_out
);
```

```
//>>>>>>>>>>>>>>>>>>>> ROM SETTING <<<<<<<<<<<<<<<<<<<<//
//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> background
// logic [8:0] background_row_addr; // no need for address
// logic [9:0] background_col_addr; // no need for address
wire [11:0] rom_background_data;
rom_background_controller rom_background_controller_inst(
    .clk(clk),
    .row_addr(row_addr),
    .col_addr(col_addr),
    .data(rom_background_data)
);
//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> pacman
wire [1:0] pacman_number; // 0 1 2 3 2 1 for a period
cyclic_4_counter c4c_inst(
    .v_sync(v_sync),
    .rst(rst),
    .count(pacman_number)
);
logic [4:0] rom_pacman_row_addr, rom_pacman_col_addr;
wire [11:0] rom_pacman_data;
rom_pacman_controller rom_pacman_controller_inst(
    .clk(clk),
    .direction(pacman_direction),
    .number(pacman_number),
    .row_addr(rom_pacman_row_addr),
    .col_addr(rom_pacman_col_addr),
    .data(rom_pacman_data)
);
//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> life icon
logic [4:0] rom_life_row_addr, rom_life_col_addr;
wire [11:0] rom_life_data;
rom_pacman_controller life_icon_inst(
    .clk(clk),
    .direction(2'b11),
    .number(2'b10),
    .row_addr(rom_life_row_addr),
    .col_addr(rom_life_col_addr),
    .data(rom_life_data)
);
//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> ghost
wire ghost_number;
cyclic_2_counter c2c_inst(
    .v_sync(v_sync),
    .rst(rst),
    .count(ghost_number)
);
logic [4:0] rom_ghost_row_addr, rom_ghost_col_addr; // no need for address
wire [11:0] rom_ghost_data; // no need for address
rom_ghost_controller rom_ghost_controller_inst(
    .clk(clk),
    .direction(ghost_direction),
    .number(ghost_number),
    .row_addr(rom_ghost_row_addr),
    .col_addr(rom_ghost_col_addr),
    .data(rom_ghost_data)
);
//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> numbers
wire [11:0] rom_numbers_data;
logic [4:0] rom_numbers_row_addr, rom_numbers_col_addr;
logic [3:0] rom_numbers_number; // 0 to 9
rom_numbers_controller rom_numbers_controller_inst(
    .clk(clk),
    .number(rom_numbers_number),
```

```
    .row_addr(rom_numbers_row_addr),
    .col_addr(rom_numbers_col_addr),
    .data(rom_numbers_data)
);
//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> dots
logic [4:0] rom_dot_row_addr, rom_dot_col_addr;
wire [11:0] rom_dot_data;
rom_dot_controller rom_dot_controller_inst(
    .clk(clk),
    .row_addr(rom_dot_row_addr),
    .col_addr(rom_dot_col_addr),
    .data(rom_dot_data)
);
//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> state
logic [5:0] rom_state_row_addr;
logic [7:0] rom_state_col_addr;
logic rom_state_state; // 0 for ready, 1 for gamover
wire [11:0] rom_state_data;
rom_state_controller rom_state_controller_inst(
    .clk(clk),
    .state(rom_state_state),
    .row_addr(rom_state_row_addr),
    .col_addr(rom_state_col_addr),
    .data(rom_state_data)
);


//>>>>>>>>>>>>>>>>>>>> RENDERING <<<<<<<<<<<<<<<<<<<<//
always @(*) begin
    if(rst) begin
        rom_pacman_col_addr <= 0;
        rom_pacman_row_addr <= 0;
        rom_ghost_col_addr <= 0;
        rom_ghost_row_addr <= 0;
        rom_numbers_col_addr <= 0;
        rom_numbers_row_addr <= 0;
        rom_state_row_addr <= 0;
        rom_state_col_addr <= 0;
        rom_state_state <= 0;
        rom_dot_col_addr <= 0;
        rom_dot_row_addr <= 0;
        data_out <= 0;
    end else
    if(!rdn) begin
        //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> is ghost
        if(
            row_addr > 29 &&
            col_addr > 29 &&
            row_addr - 30 > ghost_y - 1 &&
            row_addr - 30 < ghost_y + 20 &&
            col_addr - 30 > ghost_x - 1 &&
            col_addr - 30 < ghost_x + 20
        ) begin
            rom_ghost_row_addr <= (row_addr - 30) - ghost_y;
            rom_ghost_col_addr <= (col_addr - 30) - ghost_x;
            data_out <= rom_ghost_data;
        //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> is pacman
        end else if(
            row_addr > 29 &&
            col_addr > 29 &&
            row_addr - 30 > pacman_y - 1 &&
            row_addr - 30 < pacman_y + 20 &&
            col_addr - 30 > pacman_x - 1 &&
            col_addr - 30 < pacman_x + 20
```

```
        ) begin
            rom_pacman_row_addr <= (row_addr - 30) - pacman_y;
            rom_pacman_col_addr <= (col_addr - 30) - pacman_x;
            data_out <= rom_pacman_data;
        //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> is dot
        end else if(
            row_addr > 29 &&
            row_addr < 450 &&
            col_addr > 29 &&
            col_addr < 450 && // in map
            map[(row_addr - 30) / 20][(col_addr - 30) / 20] == `grid_have_dot // have dot
        ) begin
            rom_dot_row_addr <= (row_addr - 30) % 20;
            rom_dot_col_addr <= (col_addr - 30) % 20;
            data_out <= rom_dot_data;
        //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> is state and have to display
        end else if(
            row_addr > 219 &&
            row_addr < 280 &&
            col_addr > 459 &&
            col_addr < 600 && // in state bar
            state != `state_idle // have to display
        ) begin
            rom_state_row_addr <= row_addr - 220;
            rom_state_col_addr <= col_addr - 460;
            data_out <= rom_state_data;
            if(state == `state_ready) rom_state_state <= 0; // show ready
            else rom_state_state <= 1; // show gameover
        //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> is number
        end else if(
            row_addr > 103 &&
            row_addr < 125 &&
            col_addr > 459 &&
            col_addr < 553 // in number bar
        ) begin
            if(col_addr > 459 && col_addr < 481) begin // first digit
                rom_numbers_row_addr <= row_addr - 104;
                rom_numbers_col_addr <= col_addr - 460;
                rom_numbers_number <= score[15:12];
                data_out <= rom_numbers_data;
            end else if(col_addr > 483 && col_addr < 505) begin // second digit
                rom_numbers_row_addr <= row_addr - 104;
                rom_numbers_col_addr <= col_addr - 484;
                rom_numbers_number <= score[11:8];
                data_out <= rom_numbers_data;
            end else if(col_addr > 507 && col_addr < 529) begin // third dight
                rom_numbers_row_addr <= row_addr - 104;
                rom_numbers_col_addr <= col_addr - 508;
                rom_numbers_number <= score[7:4];
                data_out <= rom_numbers_data;
            end else if(col_addr > 531 && col_addr < 553) begin // fourth digit
                rom_numbers_row_addr <= row_addr - 104;
                rom_numbers_col_addr <= col_addr - 532;
                rom_numbers_number <= score[3:0];
                data_out <= rom_numbers_data;
            end else begin
                data_out <= 12'b0;
            end
        //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> is life
        end else if(
            row_addr > 163 &&
            row_addr < 184 &&
            col_addr > 459 &&
            col_addr < 529 // in life bar
```

```
        ) begin
            if(col_addr > 459 && col_addr < 480 && life_count > 0) begin // first life
                rom_life_col_addr <= col_addr - 460;
                rom_life_row_addr <= row_addr - 164;
                data_out <= rom_life_data;
            end else if(col_addr > 483 && col_addr < 504 && life_count > 1) begin // second life
                rom_life_col_addr <= col_addr - 484;
                rom_life_row_addr <= row_addr - 164;
                data_out <= rom_life_data;
            end else if(col_addr > 507 && col_addr < 528 && life_count > 2) begin // third life
                rom_life_col_addr <= col_addr - 508;
                rom_life_row_addr <= row_addr - 164;
                data_out <= rom_life_data;
            end else begin
                data_out <= 12'b0;
            end
        //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> just show background
        end else begin
            data_out <= rom_background_data;
        end
    end
end


endmodule
```

6.3.5.2 `cyclic_counter`

```
`define interval 5 // 12 fps

module cyclic_4_counter( // 0 1 2 3 2 1,
    input v_sync, // frame by frame
    input rst,
    output wire [1:0] count
);

logic [1:0] array [0:5] = {
    2'b00, 2'b01, 2'b10, 2'b11, 2'b10, 2'b01
};

logic [3:0] interval_counter; // up to interval = 16
logic [2:0] count_value; // 0 to 5

assign count = array[count_value];

always @(negedge v_sync or posedge rst) begin
    if(rst) begin
        interval_counter <= 0;
        count_value <= 0;
    end else begin
        if(interval_counter == `interval - 1) begin
            interval_counter <= 0;
            count_value <= (count_value + 1) % 6;
        end else begin
            interval_counter <= interval_counter + 1;
        end
    end
end

endmodule
```

```
module cyclic_2_counter(
    input v_sync, // frame by frame
    input rst,
    output logic count
);

logic [2:0] inner_counter;

always @(posedge rst or negedge v_sync) begin
    if(rst) begin
        count <= 0;
        inner_counter <= 0;
    end else begin
        if(inner_counter == `interval - 1) begin
            inner_counter <= 0;
            count <= ~count;
        end else begin
            inner_counter <= inner_counter + 1;
        end
    end
end

endmodule
```

6.3.5.3 `rom_controllers`

```
module rom_background_controller(
    input clk,
    input wire [8:0] row_addr,
    input wire [9:0] col_addr,
    output wire [11:0] data
);

wire [18:0] rom_addr;
assign rom_addr = row_addr * 640 + col_addr;

rom_background_12_480_640 inst(
    .clka(clk),
    .addra(rom_addr),
    .douta(data)
);

endmodule

module rom_dot_controller(
    input clk,
    input wire [4:0] row_addr, col_addr,
    output wire [11:0] data
);

wire [8:0] rom_addr;
assign rom_addr = row_addr * 20 + col_addr;

rom_dot_12_20_20 inst(
    .clka(clk),
    .addra(rom_addr),
    .douta(data)
);

endmodule
```

```
module rom_ghost_controller(
    input clk, number,
    input wire [1:0] direction,
    input wire [4:0] row_addr, col_addr,
    output wire [11:0] data
);

wire [11:0] rom_addr;
assign rom_addr = direction * 800 + number * 400 + row_addr * 20 + col_addr;

rom_ghost_12_8_20_20 inst(
    .clka(clk),
    .addra(rom_addr),
    .douta(data)
);

endmodule

module rom_pacman_controller(
    input clk,
    input wire [1:0] direction, number,
    input wire [4:0] row_addr, col_addr,
    output wire [11:0] data
);

wire [12:0] rom_addr;
assign rom_addr = direction * 1600 + number * 400 + row_addr * 20 + col_addr;

rom_pacman_12_16_20_20 inst(
    .clka(clk),
    .addra(rom_addr),
    .douta(data)
);

endmodule

module rom_pacman_controller(
    input clk,
    input wire [1:0] direction, number,
    input wire [4:0] row_addr, col_addr,
    output wire [11:0] data
);

wire [12:0] rom_addr;
assign rom_addr = direction * 1600 + number * 400 + row_addr * 20 + col_addr;

rom_pacman_12_16_20_20 inst(
    .clka(clk),
    .addra(rom_addr),
    .douta(data)
);

endmodule

module rom_state_controller(
    input clk,
    input state, // 0 ready, 1 gameover, only calleed when needed
    input wire [5:0] row_addr,
    input wire [7:0] col_addr,
    output wire [11:0] data
);

wire [14:0] rom_addr;
assign rom_addr = state * 8400 + row_addr * 140 + col_addr;
```

```
rom_state_12_2_60_140 inst(
    .clka(clk),
    .addra(rom_addr),
    .douta(data)
);

endmodule
```

6.3.6 `project_consraints.xdc`

```
# Main clock
set_property PACKAGE_PIN AC18 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports clk]
create_clock -period 9.9305 -name clk [get_ports "clk"]



# buzzer
set_property PACKAGE_PIN AF25 [get_ports beep]
set_property IOSTANDARD LVCMOS33 [get_ports beep]


# VGA
set_property PACKAGE_PIN N21 [get_ports {red[0]}]
set_property PACKAGE_PIN N22 [get_ports {red[1]}]
set_property PACKAGE_PIN R21 [get_ports {red[2]}]
set_property PACKAGE_PIN P21 [get_ports {red[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {red[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {red[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {red[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {red[3]}]
set_property PACKAGE_PIN R22 [get_ports {green[0]}]
set_property PACKAGE_PIN R23 [get_ports {green[1]}]
set_property PACKAGE_PIN T24 [get_ports {green[2]}]
set_property PACKAGE_PIN T25 [get_ports {green[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {green[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {green[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {green[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {green[3]}]
set_property PACKAGE_PIN T20 [get_ports {blue[0]}]
set_property PACKAGE_PIN R20 [get_ports {blue[1]}]
set_property PACKAGE_PIN T22 [get_ports {blue[2]}]
set_property PACKAGE_PIN T23 [get_ports {blue[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {blue[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {blue[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {blue[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {blue[3]}]
set_property PACKAGE_PIN M22 [get_ports {h_sync}]
set_property IOSTANDARD LVCMOS33 [get_ports {h_sync}]
set_property PACKAGE_PIN M21 [get_ports {v_sync}]
set_property IOSTANDARD LVCMOS33 [get_ports {v_sync}]


# PS2
set_property PACKAGE_PIN N18 [get_ports PS2_clk]
set_property IOSTANDARD LVCMOS33 [get_ports PS2_clk]
set_property PACKAGE_PIN M19 [get_ports PS2_data]
set_property IOSTANDARD LVCMOS33 [get_ports PS2_data]
```

```
# Reset switch
set_property PACKAGE_PIN AA10 [get_ports {sw}]
set_property IOSTANDARD LVCMOS15 [get_ports {sw}]
```

6.4 其他代码

6.4.1 `png_convertor.py`

```python
from PIL import Image

def rgb_to_12bit_hex(rgb):
    r, g, b = (rgb[0] >> 4, rgb[1] >> 4, rgb[2] >> 4)
    return f'{r:01x}{g:01x}{b:01x}'

def convert_png_to_coe(png_path, coe_path):
    # ipen png file
    img = Image.open(png_path).convert('RGB')
    size = img.size
    pixels = list(img.getdata())


    hex_pixels = [rgb_to_12bit_hex(pixel) for pixel in pixels]

    # write
    with open(coe_path, 'w') as coe_file:
        coe_file.write('memory_initialization_radix=16;\n')
        coe_file.write('memory_initialization_vector=\n')
        for i, hex_pixel in enumerate(hex_pixels):
            if i != 0 and i % size[0] == 0:
                coe_file.write(',\n')
            else:
                if i != 0:
                    coe_file.write(', ')
            coe_file.write(hex_pixel)
        coe_file.write(';\n')

convert_png_to_coe('image.png', 'image.coe')
```